3/5/1 (Item 1 from file: 351)
DIALOG(R)File 351:Derwent WPI

015309997 **Image available**
WPI Acc No: 2003-370931/200335
XRPX Acc No: N03-295810

Multi-thread executing method in parallel processing system, involves completing thread process by executing specific fork instruction based on predetermined condition

Patent Assignee: NEC CORP (NIDE)
Inventor: MATSUSHITA S; OHSAWA T
Number of Countries: 003 Number of Patents: 003
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
US 20030014473	A1	20030116	US 2002189455	A	20020708	200335	B
JP 2003029985	A	20030131	JP 2001212247	A	20010712	200335	
GB 2380573	A	20030409	GB 200216275	A	20020712	200335	

Priority Applications (No Type Date): JP 2001212247 A 20010712
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
US 20030014473	A1		66	G06F-009/00	
JP 2003029985	A		38	G06F-009/46	
GB 2380573	A			G06F-009/38	

Abstract (Basic): US 20030014473 A1

NOVELTY - A specific fork instruction is selected for creating an effective child thread during execution of parent thread. The processing of thread is completed by executing the instruction upto an address just before a starting address of the effective child thread.

DETAILED DESCRIPTION - An INDEPENDENT CLAIM is included for parallel processing system.

USE - For performing multi-thread execution in parallel processing system (claimed).

ADVANTAGE - The system is capable of assuring the Fork-Once limitation at a time of program execution, thereby increasing the probability of the fork with improved performance.

DESCRIPTION OF DRAWING(S) - The figure shows a flowchart explaining the multi-thread executing process.

pp; 66 DwgNo 4/39

Title Terms: MULTI; THREAD; EXECUTE; METHOD; PARALLEL; PROCESS; SYSTEM; COMPLETE; THREAD; PROCESS; EXECUTE; SPECIFIC; FORK; INSTRUCTION; BASED; PREDETERMINED; CONDITION
Derwent Class: T01
International Patent Class (Main): G06F-009/00; G06F-009/38; G06F-009/46
International Patent Class (Additional): G06F-015/16; G06F-015/177
File Segment: EPI

3/5/2 (Item 1 from file: 347)
DIALOG(R)File 347:JAPIO

07536150 **Image available**
MULTITHREAD EXECUTION METHOD AND PARALLEL PROCESSOR SYSTEM

PUB. NO.: 2003-029985 A]
PUBLISHED: January 31, 2003 (20030131)
INVENTOR(s): OSAWA HIROSHI
MATSUSHITA SATOSHI
APPLICANT(s): NEC CORP
APPL. NO.: 2001-212247 [JP 2001212247]
FILED: July 12, 2001 (20010712)
INTL CLASS: G06F-009/46; G06F-015/16; G06F-015/177

ABSTRACT

PROBLEM TO BE SOLVED: To guarantee fork one time limitation at the time of executing a parallelized program.

SOLUTION: In the multithread execution method of dividing a single program into a plurality of threads and parallelly executing them in a plurality of processors, a program PE1 guarantees the fork one time limitation at the time of program execution by selecting one fork instruction generating a valid slave thread from a plurality of the fork instructions present in a master thread while executing the master thread by canceling (c, f) the slave thread in the case that the slave thread generated from the thread already exists or invalidating (d, g) all the fork instructions other than the fork instruction fork which succeeds in forking the slave thread first for each fork instruction fork of the thread during execution. The processor PE1 which generates the valid slave thread ends the processing of the thread by completing the execution to the instruction of an address immediately before the start address of the valid slave thread.

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2003－29985
（P2003－29985A）
(43)公開日　平成15年１月31日(2003.1.31)

(51)Int.Cl.⁷	識別記号	Ｆ Ｉ	テーマコード（参考）
Ｇ０６Ｆ　9/46	３６０	Ｇ０６Ｆ　9/46	３６０Ｂ　５Ｂ０４５
15/16	６１０	15/16	６１０Ｚ　５Ｂ０９８
15/177	６７４	15/177	６７４Ａ
	６８１		６８１Ｂ

審査請求　有　　請求項の数32　ＯＬ　（全 38 頁）

(21)出願番号　特願2001－212247(P2001－212247)

(22)出願日　平成13年７月12日(2001.7.12)

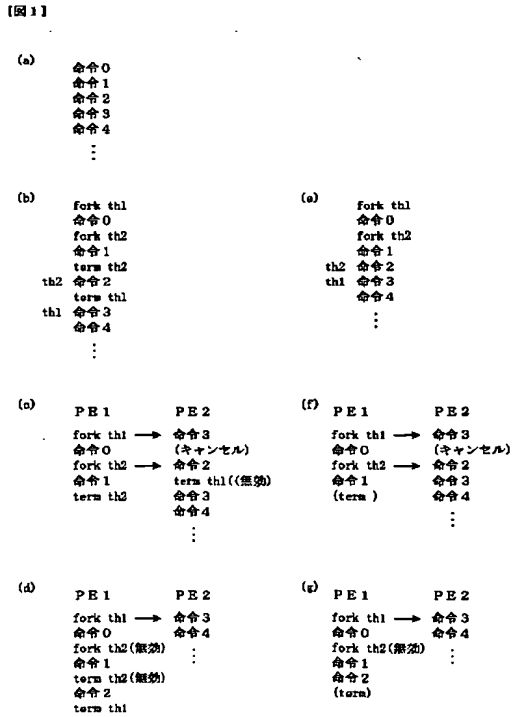
(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号
(72)発明者　大澤　拓
東京都港区芝五丁目７番１号　日本電気株式会社内
(72)発明者　松下　智
東京都港区芝五丁目７番１号　日本電気株式会社内
(74)代理人　100088959
弁理士　境　廣巳

Ｆターム(参考)　5B045 GG11
5B098 AA10 GA05 GC14

(54)【発明の名称】　マルチスレッド実行方法及び並列プロセッサシステム

(57)【要約】
【課題】　フォーク１回制限を並列化プログラムの実行時に保証する。
【解決手段】　単一のプログラムを複数のスレッドに分割し複数のプロセッサで並列に実行するマルチスレッド実行方法において、プログラムPE1 は、実行中のスレッドのフォーク命令fork毎に、当該スレッドから生成された子スレッドが既に存在する場合にはその子スレッドをキャンセルするか（ｃ，ｆ）、最初に子スレッドのフォークに成功したフォーク命令fork以外の全てのフォーク命令を無効化することで（ｄ，ｇ）、親スレッド中に存在する複数のフォーク命令のうちから有効な子スレッドを生成する１つのフォーク命令を親スレッド実行中に選択することによりフォーク１回制限をプログラム実行時に保証する。有効な子スレッドを生成したプロセッサPE1 は、有効な子スレッドの開始アドレスの直前のアドレスの命令まで実行を完了することによりスレッドの処理を終了する。

【図１】



1

【特許請求の範囲】

【請求項１】　単一のプログラムを複数のスレッドに分割し複数のプロセッサで並列に実行するマルチスレッド実行方法において、

各々の前記プロセッサは、親スレッド中に存在する複数のフォーク命令のうちから有効な子スレッドを生成する１つのフォーク命令を親スレッド実行中に選択することによりフォーク１回制限をプログラム実行時に保証し、且つ、有効な子スレッドを生成したプロセッサは、有効な子スレッドの開始アドレスの直前のアドレスの命令まで実行を完了することによりスレッドの処理を終了することを特徴とするマルチスレッド実行方法。

【請求項２】　親スレッドのフォーク命令毎に前記親スレッドから生成された子スレッドが既に存在する場合にはその子スレッドをキャンセルする請求項１記載のマルチスレッド実行方法。

【請求項３】　親スレッドの実行を開始したプロセッサで最初に子スレッドのフォークに成功したフォーク命令以外の全てのフォーク命令を無効化する請求項１記載のマルチスレッド実行方法。

【請求項４】　前記プロセッサは、各フォーク命令に対応して並列化プログラム中のフォーク先アドレスの直前に挿入されているターム命令のうち、有効な子スレッドをフォークしたフォーク命令に対応するターム命令によってスレッドの処理を終了する請求項１乃至３の何れか１項に記載のマルチスレッド実行方法。

【請求項５】　前記プロセッサは、プログラムカウンタの値が有効な子スレッドの開始アドレスと一致したときにスレッドの処理を終了する請求項１乃至３の何れか１項に記載のマルチスレッド実行方法。

【請求項６】　親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在しなかった場合、子スレッドの実行を開始できる他のプロセッサが生じるまで前記フォーク命令の実行をウエイトする請求項１乃至３の何れか１項に記載のマルチスレッド実行方法。

【請求項７】　親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在しなかった場合、親スレッドのレジスタファイルの内容を退避させ、子スレッドの実行を開始できる他のプロセッサが生じた時点で前記退避した情報に基づいて子スレッドのフォークを行う請求項１乃至３の何れか１項に記載のマルチスレッド実行方法。

【請求項８】　親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在しなかったフォーク命令は無効化する請求項１乃至３の何れか１項に記載のマルチスレッド実行方法。

【請求項９】　親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在せず且つ親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他のプロセッサが生じなかったフォーク命令は無効化する請求項１乃至３の何れか１項に記載のマルチスレッド実行方法。

2

【請求項１０】　親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在せず且つ親スレッドのレジスタファイルのレジスタのうち子スレッドに継承すべきレジスタが更新される前に子スレッドの実行を開始できる他のプロセッサが生じなかったフォーク命令は無効化する請求項１乃至３の何れか１項に記載のマルチスレッド実行方法。

【請求項１１】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行し且つスレッド実行のキャンセルが可能な仮実行用バッファを有する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、前記親スレッドを実行しているプロセッサのフォーク先プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）前記プロセッサに設けたフォークドビットをリセットした状態で前記プロセッサに親スレッドの実行を開始させるステップ、（ｂ）親スレッド中の各フォーク命令の時点で前記プロセッサに設けたレジスタにフォーク先アドレスを保存し、前記フォークドビットがリセットされている場合は、前記プロセッサのフォーク先プロセッサがフリー状態のときは前記子スレッドをフォーク先プロセッサにフォークして前記フォークドビットをセットし、フォーク先プロセッサがビジー状態のときは前記フォーク命令を無効化し、前記フォークドビットがセットされている場合は、フォーク先プロセッサのスレッド実行をキャンセルして前記子スレッドをフォーク先プロセッサにフォークするステップ、（ｃ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１２】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行し且つスレッド実行のキャンセルが可能な仮実行用バッファを有する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、前記親ス

レッドを実行しているプロセッサのフォーク先プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）前記プロセッサに設けたフォークドビット及びフォーク有効ビットをリセットした状態で前記プロセッサに親スレッドの実行を開始させるステップ、（ｂ）親スレッド中の各フォーク命令の時点で前記プロセッサに設けたレジスタにフォーク先アドレスを保存すると共に前記フォーク有効ビットをセットするステップ、（ｃ）親スレッドのレジスタファイルが更新されたときに前記フォーク有効ビットをリセットするステップ、（ｄ）前記プロセッサの前記フォーク有効ビットがセットされている場合、前記フォークドビットがリセットされているときは前記プロセッサのフォーク先プロセッサがフリー状態になった時点で前記子スレッドをフォーク先プロセッサにフォークして前記フォークドビットをセットすると共に前記フォーク有効ビットをリセットし、前記フォークドビットがセットされているときはフォーク先プロセッサのスレッド実行をキャンセルして前記子スレッドをフォーク先プロセッサにフォークするステップ、（ｅ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１３】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行し且つスレッド実行のキャンセルが可能な仮実行用バッファを有する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、前記親スレッドを実行しているプロセッサのフォーク先プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）前記プロセッサに設けたフォークドビット及びフォーク有効ビットをリセットした状態で前記プロセッサに親スレッドの実行を開始させるステップ、（ｂ）親スレッド中の各フォーク命令の時点で前記プロセッサに設けたレジスタにフォーク先アドレスを保存すると共に前記フォーク有効ビットをセットするステップ、（ｃ）親スレッドのレジスタファイルのレジスタの内、子ステップに継承すべきレジスタが更新されたときに前記フォーク有効ビットをリセットするステップ、（ｄ）前記プロセッサの前記フォーク有効ビットがセットされている場合、前記フォークドビットがリセットされているときは前記プロセッサのフォーク先プロセッサがフリー状態になった時点で前記子スレッドをフォーク先プロセッサにフォークして前記フォークドビットをセットすると共に前記フォーク有効ビットをリセットし、前記フォークドビットがセットされているときはフォーク先プロセッサのスレッド実行をキャンセルして前記子スレッドをフォーク先プロセッサにフォークするステップ、（ｅ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１４】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行し且つスレッド実行のキャンセルが可能な仮実行用バッファを有する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、前記親スレッドを実行しているプロセッサのフォーク先プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）前記プロセッサに設けたフォークドビットをリセットした状態で前記プロセッサに親スレッドの実行を開始させるステップ、（ｂ）親スレッド中の各フォーク命令の時点で前記プロセッサに設けたレジスタにフォーク先アドレスを保存し、前記フォークドビットがリセットされている場合は、前記プロセッサのフォーク先プロセッサがフリー状態であるときは直ちに、フリー状態でないときはフリー状態になるまで前記フォーク命令をウエイトしてから前記子スレッドをフォーク先プロセッサにフォークして前記フォークドビットをセットし、前記フォークドビットがセットされている場合は、フォーク先プロセッサのスレッド実行をキャンセルして前記子スレッドをフォーク先プロセッサにフォークするステップ、（ｃ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１５】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行し且つスレッド実行のキャンセルが可能な仮実行用バッファを有する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジ

スタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、前記親スレッドを実行しているプロセッサのフォーク先プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）前記プロセッサに設けたフォークドビットをリセットした状態で前記プロセッサに親スレッドの実行を開始させるステップ、（ｂ）親スレッド中の各フォーク命令の時点で前記プロセッサに設けたレジスタにフォーク先アドレスを保存し、前記フォークドビットがリセットされている場合は、前記プロセッサのフォーク先プロセッサがフリー状態のときは直ちに前記レジスタファイルの内容に基づいて前記子スレッドをフォーク先プロセッサにフォークして前記フォークドビットをセットし、前記プロセッサのフォーク先プロセッサがビジー状態のときは前記レジスタファイルの内容を前記プロセッサに設けた退避バッファに退避してフォーク先プロセッサがフリー状態になるまで子スレッドのフォークを保留し、前記フォークドビットがセットされている場合は、フォーク先プロセッサのスレッド実行をキャンセルして前記子スレッドをフォーク先プロセッサにフォークするステップ、（ｃ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１６】　請求項１１乃至１５の何れか１項に記載のマルチスレッド実行方法において、前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップの代わりに、前記プロセッサは、各フォーク命令に対応して並列化プログラム中のフォーク先アドレスの直前に挿入されているターム命令のうち、有効な子スレッドをフォークしたフォーク命令に対応するターム命令によってスレッドの処理を終了するステップを含むマルチスレッド実行方法。

【請求項１７】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、前記親スレッドを実行しているプロセッサのフォーク先プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）前記プロセッサに設けたフォークドビットをリセットした状態で前記プロセッサに親スレッドの実行を開始させるステップ、（ｂ）親スレッド中のフォーク命令の時点で、前記フォークドビットがリセットされている場合は、前記親スレッドを実行しているプロセッサのフォーク先プロセッサがフリー状態のときは前記子スレッドをフォークして前記フォークドビットをセットすると共に前記プロセッサに設けたレジスタにフォーク先アドレスを保存し、フォーク先プロセッサがビジー状態のときは前記フォーク命令を無効化し、前記フォークドビットがセットされている場合は、前記フォーク命令を無効化するステップ、（ｃ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１８】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、前記親スレッドを実行しているプロセッサのフォーク先プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）前記プロセッサに設けたフォークドビットをリセットした状態で前記プロセッサに親スレッドの実行を開始させるステップ、（ｂ）親スレッド中のフォーク命令の時点で、前記フォークドビットがリセットされている場合は、前記プロセッサに設けたフォーク有効ビットをセットすると共に前記プロセッサに設けたレジスタにフォーク先アドレスを保存し、前記フォークドビットがセットされている場合は、前記フォーク命令を無効化するステップ、（ｃ）親スレッドのレジスタファイルが更新されたときに前記フォーク有効ビットをリセットするステップ、（ｄ）前記プロセッサの前記フォーク有効ビットがセットされている場合、前記親スレッドを実行しているプロセッサのフォーク先プロセッサがフリー状態であれば前記プロセッサの前記レジスタに保存されたフォーク先アドレスから始まる子スレッドをフォーク先プロセッサにフォークして前記プロセッサの前記フォークドビットをセットし且つ前記フォーク有効ビットをリセットするステップ、（ｅ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１９】　各々プログラムカウンタ及びレジスタ

ファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、前記親スレッドを実行しているプロセッサのフォーク先プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）前記プロセッサに設けたフォークドビットをリセットした状態で前記プロセッサに親スレッドの実行を開始させるステップ、（ｂ）親スレッド中のフォーク命令の時点で、前記フォークドビットがリセットされている場合は前記プロセッサに設けたフォーク有効ビットをセットすると共に前記プロセッサに設けたレジスタにフォーク先アドレスを保存し、前記フォークドビットがセットされている場合は前記フォーク命令を無効化するステップ、（ｃ）親スレッドのレジスタファイルのレジスタのうち子スレッドに継承すべきレジスタが更新されたときに前記フォーク有効ビットをリセットするステップ、（ｄ）前記フォーク有効ビットがセットされている場合、前記親スレッドを実行しているプロセッサのフォーク先プロセッサがフリー状態であれば前記子スレッドをフォーク先プロセッサにフォークして前記フォークドビットをセットし且つ前記フォーク有効ビットをリセットするステップ、（ｅ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項２０】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、前記親スレッドを実行しているプロセッサのフォーク先プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）前記プロセッサに設けたフォークドビットをリセットした状態で前記プロセッサに親スレッドの実行を開始させるステップ、（ｂ）親スレッド中のフォーク命令の時点で、前記フォークドビットがリセットされている場合は、前記親スレッドを実行しているプロセッサのフォーク先プロセッサがフリー状態のときは直ちに、フリー状態でなればフォーク先プロセッサがフリー状態になるまで前記フォーク命令をウエイトしてから前記子スレッドをフォークして前記フォークドビットをセットすると共に前記プロセッサに設けたレジスタにフォーク先アドレスを保存し、前記フォークドビットがセットされている場合は、前記フォーク命令を無効化するステップ、（ｃ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項２１】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、前記親スレッドを実行しているプロセッサのフォーク先プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）前記プロセッサに設けたフォークドビットをリセットした状態で前記プロセッサに親スレッドの実行を開始させるステップ、（ｂ）親スレッド中のフォーク命令の時点で、前記フォークドビットがリセットされている場合は、前記親スレッドを実行しているプロセッサのフォーク先プロセッサがフリー状態のときは直ちに前記レジスタファイルの内容に基づいて前記子スレッドを隣接プロセッサにフォークして前記フォークドビットをセットし、フォーク先プロセッサがビジー状態のときは前記レジスタファイルの内容を前記プロセッサに設けた退避バッファに退避してフォーク先プロセッサがフリー状態になるまで子スレッドのフォークを保留し、前記フォークドビットがセットされている場合は、前記フォーク命令を無効化するステップ、（ｃ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項２２】　請求項１７乃至２１の何れか１項に記載のマルチスレッド実行方法において、前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップの代わりに、前記プロセッサは、各フォーク命令に対応して並列化プログラム中のフォーク先アドレスの直前に挿入されているターム命令のうち、有効な子スレッドをフォークしたフォーク命令に対応するターム命令

によってスレッドの処理を終了するステップを含むマルチスレッド実行方法。

【請求項２３】　単一のプログラムを複数のスレッドに分割し複数のプロセッサで並列に実行する並列プロセッサシステムにおいて、各々の前記プロセッサは、親スレッド中に存在する複数のフォーク命令のうちから有効な子スレッドを生成する１つのフォーク命令を親スレッド実行中に選択することによりフォーク１回制限をプログラム実行時に保証する手段と、有効な子スレッドを生成したプロセッサは、有効な子スレッドの開始アドレスの直前のアドレスの命令まで実行を完了することによりスレッドの処理を終了する手段とを備えたことを特徴とする並列プロセッサシステム。

【請求項２４】　親スレッドのフォーク命令毎に前記親スレッドから生成された子スレッドが既に存在する場合にはその子スレッドをキャンセルする請求項２３記載の並列プロセッサシステム。

【請求項２５】　親スレッドの実行を開始したプロセッサで最初に子スレッドのフォークに成功したフォーク命令以外の全てのフォーク命令を無効化する請求項２３記載の並列プロセッサシステム。

【請求項２６】　前記プロセッサは、各フォーク命令に対応して並列化プログラム中のフォーク先アドレスの直前に挿入されているターム命令のうち、有効な子スレッドをフォークしたフォーク命令に対応するターム命令によってスレッドの処理を終了する請求項２３乃至２５の何れか１項に記載の並列プロセッサシステム。

【請求項２７】　前記プロセッサは、プログラムカウンタの値が有効な子スレッドの開始アドレスと一致したときにスレッドの処理を終了する請求項２３乃至２５の何れか１項に記載の並列プロセッサシステム。

【請求項２８】　親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在しなかった場合、子スレッドの実行を開始できる他のプロセッサが生じるまで前記フォーク命令の実行をウエイトする請求項２３乃至２５の何れか１項に記載の並列プロセッサシステム。

【請求項２９】　親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在しなかった場合、親スレッドのレジスタファイルの内容を退避させ、子スレッドの実行を開始できる他のプロセッサが生じた時点で前記退避した情報に基づいて子スレッドのフォークを行う請求項２３乃至２５の何れか１項に記載の並列プロセッサシステム。

【請求項３０】　親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在しなかったフォーク命令は無効化する請求項２３乃至２５の何れか１項に記載の並列プロセッサシステム。

【請求項３１】　親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在せず且つ親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他のプロセッサが生じなかったフォーク命令は無効化する請求項２３乃至２５の何れか１項に記載の並列プロセッサシステム。

【請求項３２】　親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在せず且つ親スレッドのレジスタファイルのレジスタのうち子スレッドに継承すべきレジスタが更新される前に子スレッドの実行を開始できる他のプロセッサが生じなかったフォーク命令は無効化する請求項２３乃至２５の何れか１項に記載の並列プロセッサシステム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は並列プロセッサシステムにおけるプログラム並列実行方法に関し、より具体的には単一のプログラムを複数のスレッドに分割して複数のプロセッサにより並列に実行するマルチスレッド実行方法及び並列プロセッサシステムに関する。

【０００２】

【従来の技術】単一のプログラムを並列プロセッサシステムで並列に処理する手法として、プログラムをスレッドと呼ぶ命令流に分割して複数のプロセッサで並列に実行するマルチスレッド実行方法があり、この方法を記載した文献として、特開平１０－２７１０８号公報（以下、文献１と称す）、「Ｏｎ　Ｃｈｉｐ　Ｍｕｌｔｉｐｒｏｃｅｓｓｏｒ指向　制御並列アーキテクチャＭＵＳＣＡＴの提案」（並列処理シンポジュウムＪＳＰＰ９７論文集、情報処理学会、ｐｐ．２２９－２３６、Ｍａｙ　１９９７）（以下、文献２と称す）、特開平１０－７８８８０号公報（以下、文献３と称す）等がある。以下、これらの従来文献に記載されたマルチスレッド実行方法について説明する。

【０００３】一般にマルチスレッド実行方法において、他のプロセッサ上に新たなスレッドを生成することを、スレッドをフォーク（ｆｏｒｋ）すると言い、フォーク動作を行った側のスレッドを親スレッド、生成された新しいスレッドを子スレッド、スレッドをフォークする箇所をフォーク点、子スレッドの先頭箇所をフォーク先アドレスまたは子スレッドの開始点と呼ぶ。文献１～３では、スレッドのフォークを指示するためにフォーク点にフォーク命令が挿入される。フォーク命令にはフォーク先アドレスが指定され、フォーク命令の実行によりそのフォーク先アドレスから始まる子スレッドが他プロセッサ上に生成され、子スレッドの実行が開始される。また、スレッドの処理を終了させるターム（ｔｅｒｍ）命令と呼ばれる命令が用意されており、各プロセッサはターム命令を実行することによりスレッドの処理を終了する。

【０００４】図３７に従来のマルチスレッド実行方法の処理の概要を示す。同図（ａ）は３つのスレッドＡ、Ｂ、Ｃに分割された単一のプログラムを示す。このプログラムを単一のプロセッサで処理する場合、同図（ｂ）に示すように１つのプロセッサＰＥがスレッドＡ、Ｂ、Ｃを順番に処理していく。これに対して文献１～３のマルチスレッド実行方法では、同図（ｃ）に示すように、１つのプロセッサＰＥ１にスレッドＡを実行させ、プロセッサＰＥ１でスレッドＡを実行している最中に、スレッドＡに埋め込まれたフォーク命令によってスレッドＢを他のプロセッサＰＥ２に生成し、プロセッサＰＥ２においてスレッドＢを実行させる。また、プロセッサＰＥ２はスレッドＢに埋め込まれたフォーク命令によってスレッドＣをプロセッサＰＥ３に生成する。プロセッサＰＥ１、ＰＥ２はそれぞれスレッドＢ、Ｃの開始点の直前に埋め込まれたターム命令によってスレッドの処理を終了し、プロセッサＰＥ３はスレッドＣの最後の命令を実行すると、その次の命令（一般にはシステムコール命令）を実行する。このように複数のプロセッサでスレッドを同時に並行して実行することにより、逐次処理に比べて性能の向上が図られる。

【０００５】従来の他のマルチスレッド実行方法として、図３７（ｄ）に示すように、スレッドＡを実行しているプロセッサＰＥ１からフォークを複数回行うことにより、プロセッサＰＥ２にスレッドＢを、またプロセッサＰＥ３にスレッドＣをそれぞれ生成するマルチスレッド実行方法も存在する。この図３７（ｄ）のモデルに対して、同図（ｃ）に示したようにスレッドはその生存中に高々１回に限って有効な子スレッドを生成することができるという制約を課したマルチスレッド実行方法をフォーク１回モデルと呼ぶ。フォーク１回モデルでは、スレッド管理の大幅な簡略化が可能となり、現実的なハードウェア規模でスレッド管理部のハードウェア化が実現できる。また、個々のプロセッサは子スレッドを生成する他プロセッサが１プロセッサに限定されるため、隣接するプロセッサを単方向にリング状に接続した並列プロセッサシステムでマルチスレッド実行が可能となる。本発明はこのようなフォーク１回モデルを前提とする。

【０００６】ここで、フォーク命令時、子スレッドを生成できる空きのプロセッサが存在しない場合、従来は次の２通りの方法の何れかを採用している。

（１）親スレッドを実行しているプロセッサは、子スレッドを生成できる空きのプロセッサが生じるまで、フォーク命令の実行をウエイトする。

（２）親スレッドを実行しているプロセッサは、フォーク先アドレス及びフォーク点におけるレジスタファイルの内容を裏面の物理レジスタに保存して親スレッドの後続処理を続行する。裏面の物理レジスタに保存されたフォーク先アドレス及びレジスタファイルの内容は、子スレッドを生成できる空きのプロセッサが生じた時点で参照され、子スレッドが生成される。

【０００７】親スレッドが子スレッドを生成し、子スレッドに所定の処理を行わせるには、親スレッドのフォーク点におけるレジスタファイル中のレジスタのうち少なくとも子スレッドで必要なレジスタの値を親スレッドから子スレッドに引き渡す必要がある。このスレッド間のデータ引き渡しコストを削減するために、文献２及び３では、スレッド生成時のレジスタ値継承機構をハードウェア的に備えている。これは、スレッド生成時に親スレッドのレジスタファイルの内容を子スレッドに全てコピーするものである。子スレッド生成後は、親スレッドと子スレッドのレジスタ値の変更は独立となり、レジスタを用いたスレッド間のデータの引き渡しは行われない。スレッド間のデータ引き渡しに関する他の従来技術としては、レジスタの値を命令によりレジスタ単位で個別に転送する機構を備えた並列プロセッサシステムも提案されている。

【０００８】マルチスレッド実行方法では、実行の確定した先行スレッドを並列に実行することを基本とするが、実際のプログラムでは実行の確定するスレッドが充分に得られない場合も多い。また、動的に決定される依存やコンパイラ解析能力の限界等により並列化率が低く抑えられ所望の性能が得られない可能性が生じる。このため文献１では、制御投機を導入し、ハードウェア的にスレッドの投機実行をサポートしている。制御投機では、実行する可能性の高いスレッドを実行確定前に投機的に実行する。投機状態のスレッドは、実行の取り消しがハードウェア上可能である範囲内で仮実行を行う。子スレッドが仮実行を行っている状態を仮実行状態と言い、子スレッドが仮実行状態にあるとき親スレッドはスレッド仮生成状態にあると言う。仮実行状態の子スレッドでは共有メモリ及びキャッシュメモリへの書き込みは抑制され、別途設けた仮実行用バッファ（ｔｅｍｐｏｒａｒｙｂｕｆｆｅｒ）に対して書き込みが行われる。投機が正しいことが確定すると、親スレッドから子スレッドに対して投機成功通知が出され、子スレッドは仮実行用バッファの内容を共有メモリ及びキャッシュメモリに反映し、仮実行用バッファを用いない通常の状態となる。また親スレッドはスレッド仮生成状態からスレッド生成状態となる。他方、投機が失敗したことが確定すると、親スレッドでスレッド破棄命令（ａｂｏｒｔ）が実行され、子スレッド以下の実行がキャンセルされる。また、親スレッドはスレッド仮生成状態からスレッド未生成状態となり、再び子スレッドの生成が可能にある。つまり、フォーク１回モデルではスレッド生成は高々１回に限定されるが、制御投機を行い、投機が失敗した場合には再びフォークが可能となる。この場合においても、有効な子スレッドは高々１つである。

【０００９】その他、文献２に記載のＭＵＳＣＡＴでは、スレッド間の同期命令など、スレッドの並列動作を

柔軟に制御するための専用命令が数多く用意されている。

【００１０】

【発明が解決しようとする課題】スレッドはその生存中に高々１回に限って有効な子スレッドを生成するというフォーク１回モデルのマルチスレッド実行を実現するために、従来は文献２等に示されるように、逐次処理プログラムから並列化プログラムを生成するコンパイルの段階で、全てのスレッドが有効なフォークを１回しか実行しない命令コードになるように制限していた。即ち、フォーク１回制限を並列化プログラム上において静的に保証していた。

【００１１】しかし、分割コンパイル、関数呼び出しなどの問題により、コンパイラがフォーク１回制限を守ることは難しい。従来のマルチスレッド実行方法及び並列プロセッサシステムでは、フォーク１回制限が守られていない並列化プログラムは正しく実行できない。例えば図３８に示すようなｍａｉｎ関数及びｆｕｎｃ関数を含むプログラムにおいて、同図（ａ）に示すようにｍａｉｎ関数及びｆｕｎｃ関数の双方にフォーク命令が挿入されていると、ブロックａからブロックｂに分岐する制御フローが実行される場合にはフォーク１回制限は守られるが、ブロックａからブロックｃに分岐する制御フローが実行されると、同じスレッドからフォークが２回行われるため、フォーク１回制限が保証されず正常な実行が行えない。このため、従来はコンパイルの段階でｍａｉｎ関数かｆｕｎｃ関数の何れか一方にのみフォーク命令を挿入することで、フォーク１回制限を保証する必要があった。図３.８（ｂ）に、ｆｕｎｃ関数にのみフォーク命令を挿入し、ｍａｉｎ関数におけるブロックａの時点におけるブロックｄの先行実行を断念した並列化プログラムの例を示す。

【００１２】本発明はこのような事情に鑑みて提案されたものであり、その目的は、フォーク１回制限の保証のない並列化プログラムであってもフォーク１回モデルによるマルチスレッド実行が行える新規なマルチスレッド実行方法及び並列プロセッサシステムを提供することにある。

【００１３】

【課題を解決するための手段】本発明は、単一のプログラムを複数のスレッドに分割し複数のプロセッサで並列に実行するマルチスレッド実行方法において、各々の前記プロセッサは、親スレッド中に存在する複数のフォーク命令のうちから有効な子スレッドを生成する１つのフォーク命令を親スレッド実行中に選択することによりフォーク１回制限をプログラム実行時に保証し、且つ、有効な子スレッドを生成したプロセッサは、有効な子スレッドの開始アドレスの直前のアドレスの命令まで実行を完了することによりスレッドの処理を終了することを基本とする。

【００１４】より具体的には、第１の発明は、親スレッドのフォーク命令毎に当該親スレッドから生成された子スレッドが既に存在する場合にはその子スレッドをキャンセルすることで、親スレッド中に存在する複数のフォーク命令のうちから有効な子スレッドを生成する１つのフォーク命令を親スレッド実行中に選択し、フォーク１回制限をプログラム実行時に保証する。

【００１５】また、第２の発明は、親スレッドの実行を開始したプロセッサで最初に子スレッドのフォークに成功したフォーク命令以外の全てのフォーク命令を無効化することで、親スレッド中に存在する複数のフォーク命令のうちから有効な子スレッドを生成する１つのフォーク命令を親スレッド実行中に選択し、フォーク１回制限をプログラム実行時に保証する。

【００１６】有効な子スレッドを生成したプロセッサに、有効な子スレッドの開始アドレスの直前のアドレスの命令まで実行を完了することによりスレッドの処理を終了させるには、プロセッサが各フォーク命令に対応して並列化プログラム中のフォーク先アドレスの直前に挿入されているターム命令のうち、有効な子スレッドをフォークしたフォーク命令に対応するターム命令によってスレッドの処理を終了するように構成しても良いし、各プロセッサのプログラムカウンタの値が有効な子スレッドの開始アドレスと一致したときにスレッドの処理を終了するように構成しても良い。

【００１７】また、親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在しなかった場合、文献１に記載されるように、子スレッドの実行を開始できる他のプロセッサが生じるまでフォーク命令の実行をウエイトしたり、レジスタファイルの内容を退避させて子スレッドの実行を開始できる他のプロセッサが生じた時点で前記退避した情報に基づいて子スレッドのフォークを行うようにしても良い。更に、親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在しなかったフォーク命令は無効化したり、親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在せず且つ親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他のプロセッサが生じなかったフォーク命令は無効化したり、親スレッドの実行開始後、フォーク命令の時点で子スレッドの実行を開始できる他のプロセッサが存在せず且つ親スレッドのレジスタファイルのレジスタのうち子スレッドに継承すべきレジスタが更新される前に子スレッドの実行を開始できる他のプロセッサが生じなかったフォーク命令は無効化したりする方法を採用することも可能である。

【００１８】

【作用】本発明にあっては、各プロセッサは、親スレッドのフォーク命令毎に当該親スレッドから生成された子

スレッドが既に存在する場合にはその子スレッドをキャンセルし（第１の発明）、親スレッドの実行を開始したプロセッサで最初に子スレッドのフォークに成功したフォーク命令以外の全てのフォーク命令を無効化することで（第２の発明）、親スレッド中に存在する複数のフォーク命令のうちから有効な子スレッドを生成する１つのフォーク命令を親スレッド実行中に選択するため、フォーク１回制限の保証のない並列化プログラムであってもフォーク１回制限をプログラム実行時に保証することができ、また、有効な子スレッドを生成したプロセッサは、有効な子スレッドの開始アドレスの直前のアドレスの命令まで実行を完了することによりスレッドの処理を終了することにより、プログラムの処理を支障なく遂行できる。

【００１９】次に本発明の作用を、その理解を容易にするために簡略化したプログラム例に基づいて説明する。図１（ａ）は逐次処理プログラムの例を示し、同図（ｂ）はこの逐次処理プログラムから生成された並列化プログラムの例を示す。この例の並列化プログラム中、ｆｏｒｋ　ｔｈｉは、アドレスｔｈｉ以降の命令を子スレッドとしてフォークすることを指示するフォーク命令を示し、ｔｅｒｍ　ｔｈｉはｆｏｒｋ　ｔｈｉに対応するターム命令を示す。同図（ｂ）の並列化プログラムでは、１つのスレッドから複数回フォークを行うように記述されており、フォーク１回制限が保証されていない。

【００２０】図１（ｃ）は同図（ｂ）の並列化プログラムを第１の発明で実行したシーケンスを示す。プロセッサＰＥ１はフォーク命令ｆｏｒｋ　ｔｈ１によりプロセッサＰＥ２に子スレッドをフォークし、プロセッサＰＥ２は命令３から子スレッドの実行を開始する。プロセッサＰＥ１は引き続き命令０を実行し、再びフォーク命令ｆｏｒｋ　ｔｈ２が現れたので、プロセッサＰＥ２に子スレッドをフォークする。このときプロセッサＰＥ２上の実行中のスレッドはキャンセルされ、プロセッサＰＥ２は命令２から新たな子スレッドの実行を開始する。プロセッサＰＥ１は引き続き命令１を実行し、次の命令は有効な子スレッドをフォークしたフォーク命令ｆｏｒｋ　ｔｈ２に対応するターム命令ｔｅｒｍ　ｔｈ２なので、スレッドの処理を終了する。他方、プロセッサＰＥ２は、実行中のスレッドから未だ有効な子スレッドをフォークしていないので、命令２の次のターム命令ｔｅｒｍｔｈ１は無効化し、命令３、命令４と実行を進める。プロセッサＰＥ１で実行されるスレッドからは有効なスレッドは高々１回しかフォークしていないため、フォーク１回制限がプログラム実行時に保証されている。また、最終的に、プロセッサＰＥ１では命令０、命令１がその順に、プロセッサＰＥ２では命令２、命令３、命令４がその順にそれぞれ実行されており、プログラムの処理を支障なく遂行できる。

【００２１】図１（ｄ）は同図（ｂ）の並列化プログラムを第２の発明で実行したシーケンスを示す。プロセッサＰＥ１はフォーク命令ｆｏｒｋ　ｔｈ１によりプロセッサＰＥ２に子スレッドをフォークし、プロセッサＰＥ２は命令３から子スレッドの実行を開始する。プロセッサＰＥ１は引き続き命令０を実行し、再びフォーク命令ｆｏｒｋ　ｔｈ２が現れるが既に子スレッドを１回フォークしているのでそれを無効化し、次の命令１を実行する。更に、プロセッサＰＥ１はターム命令ｔｅｒｍ　ｔｈ２は有効な子スレッドをフォークしたフォーク命令ｆｏｒｋ　ｔｈ１に対応するターム命令でないので無効化し、引き続き命令２を実行し、有効な子スレッドをフォークしたフォーク命令ｆｏｒｋ　ｔｈ１に対応するターム命令ｔｅｒｍ　ｔｈ１を実行した時点で、スレッドの処理を終了する。プロセッサＰＥ１で実行されるスレッドからは有効なスレッドは高々１回しかフォークしていないため、フォーク１回制限がプログラム実行時に保証されている。また、最終的に、プロセッサＰＥ１では命令０、命令１、命令２がその順に、またプロセッサＰＥ２では命令３、命令４がその順にそれぞれ実行されており、プログラムの処理を支障なく遂行できる。

【００２２】図１（ｅ）は図１（ｂ）の並列化プログラム中からターム命令を取り除いた並列化プログラムを示し、図１（ｆ）は同図（ｅ）の並列化プログラムを第１の発明で実行したシーケンスを、図１（ｇ）は同図（ｅ）の並列化プログラムを第２の発明で実行したシーケンスをそれぞれ示す。ターム命令を使わない場合、各プロセッサはプログラムカウンタの値が有効な子スレッドの開始アドレスと一致したときにスレッドの処理を終了する。図１（ｆ）では、有効な子スレッドはフォーク命令ｆｏｒｋ　ｔｈ２でフォークされたスレッドであり、その開始アドレスは命令２なので、プロセッサＰＥ１はプログラムカウンタの値が命令２のアドレスと一致すると終了している。また、図１（ｇ）では、有効な子スレッドはフォーク命令ｆｏｒｋ　ｔｈ１でフォークされたスレッドであり、その開始アドレスは命令３なので、プロセッサＰＥ１はプログラムカウンタの値が命令３のアドレスと一致すると終了している。

【００２３】図１では、フォークが入れ子になっているプログラム例を用いて本発明の作用を説明したが、フォークが入れ子になっていないプログラムであっても同様にフォーク１回制限を実行時に保証し且つプログラムの処理を支障なく遂行することが可能である。

【００２４】なお、本発明においては、親スレッドから子スレッドへのフォーク時におけるレジスタの値の継承は、フォーク命令時点の親スレッドのレジスタファイルのうち少なくとも子スレッドで必要なレジスタだけを対象とすれば足りる。このための具体的なレジスタ継承機構としては、文献２及び文献３に記載されるようにスレッド生成時に親スレッドのレジスタファイルの内容すべてを子スレッドのレジスタファイルにコピーするもので

あっても良いし、レジスタ転送量の削減を図るために必要なレジスタの値だけを命令によりレジスタ単位で個別に転送するものであっても良い。

【００２５】

【発明の実施の形態】次に第１の発明の実施例について図面を参照して詳細に説明する。

【００２６】

【第１の発明の第１の実施例】図２を参照すると、本発明の並列プロセッサシステムの一例は、４スレッド並列実行型プロセッサであり、４個のプロセッサ１－ｉ（ｉ＝０～３）が信号線２－ｉによってスレッド管理部３に接続されると共に、信号線４－ｉによって共有のメモリ５に接続されている。また、プロセッサ１－ｉ相互間は通信バス６で接続されている。この例では、４スレッド並列実行型プロセッサを取り上げたが、８スレッドや１６スレッドの並列実行型プロセッサ等、一般にｎ（≧２）スレッド並列実行型プロセッサに対して本発明は適用可能である。全てのプロセッサ１－ｉ、メモリ５及びスレッド管理部３はクロックに同期して動作する。また、好ましくは、全てのプロセッサ１－ｉはメモリ５及びスレッド管理部３と共に１つの半導体チップ上に集積化される。

【００２７】各プロセッサ１－ｉは、プログラムカウンタ（以下、ＰＣと称す）及びレジスタファイルを独立に有し、ＰＣに従って、メモリ５中のスレッドの命令を同時にフェッチ、解釈、実行する機能を有している。また、各プロセッサ１－ｉは、スレッド実行の取り消し（キャンセル）が可能なように仮実行用バッファ（ｔｅｍｐｏｒａｒｙ　ｂｕｆｆｅｒ）を有している。各プロセッサ１－ｉは、スレッド管理部３から信号線２－ｉを通じてターゲットＰＣ値を伴うスレッド開始要求７ｃが送信された時点で、仮実行用バッファを使ってスレッドの実行を仮実行状態で開始する。この時点で当該プロセッサ１－ｉはビジー状態として管理される。スレッドの実行を終了するプロセッサ１－ｉは、スレッド管理部３に対して信号線２－ｉを通じてスレッド終了通知７ｄを送信する。このスレッド終了通知７ｄがスレッド管理部３で受理された時点で、当該プロセッサ１－ｉはフリー状態として管理され、プロセッサ１－ｉにスレッド終了許可７ｅが返却される。プロセッサ１－ｉはスレッド終了許可７ｅを受信した時点で仮実行状態を解き、仮実行用バッファの内容を共有のメモリ５及び図示しないキャッシュメモリに反映させ、スレッドの実行を終える。

【００２８】各プロセッサ１－ｉは、実行中の親スレッドに存在するフォーク命令によって他のプロセッサ１－ｊ（ｉ≠ｊ）に子スレッドをフォークすることができる。本実施例では、文献１～３に記載されるように、スレッド管理の簡便化のためにプロセッサ１－ｉから子スレッドをフォークできるプロセッサを、プロセッサ１－ｉの一方の隣接プロセッサ（プロセッサ１－０はプロセッサ１－１、プロセッサ１－１はプロセッサ１－２、プロセッサ１－２はプロセッサ１－３、プロセッサ１－３はプロセッサ１－０）に限定している。このようなモデルを、以下、リング型フォークモデルと称す。

【００２９】各プロセッサ１－ｉは、子スレッドのフォークを行う際、信号線２－ｉを通じてスレッド管理部３に対し、子スレッドのフォーク先アドレス（開始ＰＣ値）及び既に子スレッドを生成したことがあるか否かを示す信号（子スレッド生成済信号）を伴うフォーク要求７ａを送信する。スレッド管理部３は、フォーク要求７ａを受信すると、フォーク要求７ａに付随する子スレッド生成済信号及び隣接プロセッサの状態に基づいて、隣接する他プロセッサ１－ｊに対するフォークが可能か否かを判定し、可能ならば当該プロセッサ１－ｊに対してフォーク先アドレスを伴うスレッド開始要求７ｃを送信する一方、フォーク要求元のプロセッサ１－ｉに対しては、フォーク応答７ｂを返却する。この時点で初めてフォークが行われたことになり、フォーク応答７ｂを受信したプロセッサ１－ｉは、フォーク先のプロセッサ１－ｊのレジスタファイルに対して、親スレッドのレジスタファイルの全内容を通信バス６を通じてコピーするか、当該子スレッドで必要なレジスタの値だけをコピーすることにより、レジスタ継承を行う。

【００３０】他方、プロセッサ１－ｉからのフォーク要求時、隣接するプロセッサ１－ｊに対するフォークが不可能ならば、スレッド管理部３は、今回のフォーク要求７ａを廃棄する。これにより前記フォーク命令は無効化される。

【００３１】図３を参照すると、スレッド管理部３の一例は、スレッド管理シーケンサ１１とプロセッサ状態テーブル１２とから構成される。プロセッサ状態テーブル１２は、プロセッサ１－ｉと１対１に対応する状態エントリ１３－ｉ及び最古親ビット１４－ｉを有する。個々の状態エントリ１３－ｉは、対応するプロセッサ１－ｉがビジー状態か、フリー状態かを記録するために使用される。個々の最古親ビット１４－ｉは、対応するプロセッサ１－ｉで実行中のスレッドが全プロセッサで実行中のスレッドの最も祖先のスレッドである場合、１にセットされ、それ以外は０にセットされる。並列プログラムの実行開始時点で、最初のスレッドを実行するプロセッサに対応する最古親ビットのみが１に初期設定され、以後、プログラムのスレッドの終了、生成に応じて最古親ビット１４－ｉが適宜更新されていく。スレッド管理シーケンサ１１は、このプロセッサ状態テーブル１２を用いて各プロセッサ１－ｉにおけるスレッド生成、スレッド終了を管理する。プロセッサ１－ｉからフォーク要求７ａ、スレッド終了通知７ｄを受信した際のスレッド管理シーケンサ１１の処理例を図４及び図５に示す。

【００３２】図４を参照すると、スレッド管理シーケンサ１１は、或るクロックのタイミングで何れかのプロセ

ッサ１－ｉからフォーク要求７ａを受信すると、先ず、そのフォーク要求７ａに付随する子スレッド生成済信号を調べる（ステップＳ１）。子スレッド生成済信号が子スレッドの生成済みを示す値１でないときは（ステップＳ１でＮＯ）、隣接するプロセッサ１－ｊの状態をプロセッサ状態テーブル１２で調べ、フリー状態であれば（ステップＳ２でＹＥＳ）、フォーク可能なため、プロセッサ状態テーブル１２における当該プロセッサ１－ｊに対応するエントリ１３－ｊをフリー状態からビジー状態に更新し（ステップＳ３）、フォーク要求７ａに付随するフォーク先アドレスを添えたスレッド開始要求７ｃをフォーク先プロセッサ１－ｊに送信すると共に、要求元のプロセッサ１－ｉに対してフォーク応答７ｂを返却する（ステップＳ４）。隣接するプロセッサ１－ｊがビジー状態であれば（ステップＳ２でＮＯ）、プロセッサ１－ｊはプロセッサ１－ｉの現実行中スレッドからフォークされた子スレッド以外のスレッドを実行中であり、フォーク不可能なので、スレッド管理シーケンサ１１は、当該フォーク要求７ａを廃棄する（ステップＳ５）。

【００３３】他方、フォーク要求７ａに付随する子スレッド生成済信号が子スレッドの生成済みを示す値１のときは（ステップＳ１でＹＥＳ）、隣接プロセッサ１－ｊは当該プロセッサ１－ｉで実行中のスレッドからフォークされた子スレッドを実行していることになり、それをキャンセルすることによりフォーク可能である。このため、ステップＳ４へ進み、フォーク要求７ａに付随するフォーク先アドレスを添えたスレッド開始要求７ｃをフォーク先プロセッサ１－ｊに送信すると共に、要求元のプロセッサ１－ｉに対してフォーク応答７ｂを返却する。フォーク先プロセッサ１－ｊで実行中の子スレッドはスレッド開始要求７ｃによって後述するようにキャンセルされる。

【００３４】図５を参照すると、スレッド管理シーケンサ１１は、何れかのプロセッサ１－ｉからスレッド終了通知７ｄを受信すると、プロセッサ管理テーブル１２における当該プロセッサ１－ｉに対応する最古親ビット１４－ｉを参照し、当該プロセッサ１－ｉで実行中のスレッドが最古親スレッドであるか否か判定する（ステップＳ１１）。最古親スレッドであれば、対応する状態エントリ１３－ｉをビジー状態からフリー状態に更新すると共に、対応する最古親ビット１４－ｉを１から０に書き換え且つ隣接するプロセッサ１４－ｊの最古親ビット１４－ｊを０から１に書き換えることで、最古親スレッドを更新する（ステップＳ１２）。そして、スレッド終了通知７ｄを出したプロセッサ１－ｉに対してスレッド終了許可７ｅを送信する（ステップＳ１３）。他方、スレッド終了通知７ｄを出したプロセッサ１－ｉで実行中のスレッドが最古親スレッドでなければ（ステップＳ１１でＮＯ）、プロセッサ１－ｉで実行中のスレッドが最古親スレッドになるまでスレッド終了許可を保留する（ステップＳ１４）。

【００３５】図６を参照すると、各々のプロセッサ１－ｉは、スレッド管理部３から送信されたスレッド開始要求７ｃに付随する開始アドレス値がセットされ、その後に適宜歩進されるＰＣ２１と、ＰＣ２１に従ってメモリ５からスレッドの命令をフェッチする命令フェッチユニット２２と、フェッチされた命令をデコードし、実行する実行ユニット２３と、汎用レジスタ２４－０～２４－ｍの集合であるレジスタファイル２５と、フォーク先プロセッサに対して通信バス６経由でレジスタファイル２５の内容を転送するレジスタ転送ユニット２６と、フォーク命令実行時に実行ユニット２３からスレッド管理部３に送信されるフォーク要求７ａに付随するフォーク先アドレスを保存するレジスタ２７と、フォーク要求７ａに対するフォーク応答７ｂによってセットされるフォークドビット２８と、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスと一致するか否かを判定する一致回路２９と、フォークドビット２８及び一致回路２９の出力の論理積信号を実行ユニット２３に出力するアンドゲート３０と、仮実行用バッファ３１とを含んで構成されている。フォークドビット２８の出力は、フォーク要求７ａに付随する子スレッド生成済信号としても利用される。

【００３６】各々のプロセッサ１－ｉは、スレッド開始要求７ｃによって、それに付随する開始アドレスからスレッドの実行を開始する。スレッドを実行中であればそれをキャンセルして新たなスレッドの実行を開始する。実行ユニット２３は、スレッドの仮実行状態が解かれるまでは、実行の取り消しがハードウェア上可能である範囲内で仮実行を行う。つまり共有メモリ５及び図示しないキャッシュメモリへの書き込みは抑制し、仮実行用バッファ３１に対して書き込みを行う。スレッドの仮実行状態は、本実施例では、スレッド終了通知７ｄに対する応答としてスレッド終了許可７ｅを受信したときに解かれる。スレッドの仮実行状態が解かれると、実行ユニット２３は仮実行用バッファ３１の内容を共有メモリ５及び図示しないキャッシュメモリに反映する。仮実行用バッファ３１を用いない通常の状態となる。

【００３７】従来のマルチスレッド実行方法では、子スレッドの実行をキャンセルするには、親スレッドでスレッド破棄命令（ａｂｏｒｔ）を実行する必要があったが、本実施例のプロセッサ１－ｉではスレッド開始要求７ｃが仮実行用バッファ３１にキャンセル信号として与えられているため、新しい子スレッドを起動すれば既に実行中の子スレッドが自動的にキャンセルされるようになっており、親スレッドでスレッド破棄命令を実行する必要はない。また、本実施例では、スレッド終了許可７ｅを受信したときに仮実行状態を解除するようにしたため、従来のマルチスレッド実行方法におけるような投機

成功を子スレッドに通知する命令は必要ではない。

【００３８】また、実行ユニット２３は、フォークドビット２８がセットされている状態においてＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスと一致することによりアンドゲート３０の出力が論理“１”になると、スレッドの処理を終了すべく、スレッド管理部３に対してスレッド終了通知７ｄを送信する。ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスと一致しても、フォークドビット２８がセットされていなければ、アンドゲート３０の出力は論理“１”にならないため、実行ユニット２３はＰＣ２１に従って命令の実行を継続する。

【００３９】レジスタ転送ユニット２６は、フォークドビット２８がセットされるタイミングでフォーク先プロセッサへのレジスタ転送を開始する。レジスタ転送ユニット２６は、例えば、通信バス６のバス幅によって一度に転送できる数のレジスタ毎に、レジスタファイル２５のレジスタの値とレジスタ番号（レジスタアドレス）とをフォーク先プロセッサのレジスタファイルへ送信し、受信側のレジスタファイル２５では該当するレジスタを書き換える。

【００４０】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図７に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８がリセットされ、またスレッドを実行中であればそれがキャンセルされる（ステップＳ２１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ２２）。

【００４１】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ２４でＹＥＳ）、実行ユニット２３はフォーク先アドレスをレジスタ２７に保存（上書き）し（ステップＳ２５）、このレジスタ２７に保存したフォーク先アドレスとフォークドビット２８の値とを伴ったフォーク要求７ａをスレッド管理部３に送信する（ステップＳ２６）。スレッド管理部３は、フォークドビット２８の値が１か、０でも隣接プロセッサ１－ｊがフリー状態のときフォーク可能と判定し、要求元のプロセッサ１－ｉに対してフォーク応答７ｂを返却し、隣接プロセッサ１－ｊに対してはスレッド開始要求７ｃを送出する。フォーク応答７ｂを受信したプロセッサ１－ｉは、フォークドビット２８を１にセットし、レジスタ転送ユニット２６によって親スレッドのレジスタファイル２５の内容を通信バス６経由でフォーク先プロセッサ１－ｊのレジスタファイルに転送するレジスタ継承操作を行う（ステップＳ３０）。また、フォーク先プロセッサ１－ｊでは図７のステップＳ２１以降の処理を実行する。

【００４２】他方、スレッド管理部３は、フォークドビット２８の値が０で且つ隣接プロセッサ１－ｊがビジー状態のときフォーク不可能と判定し、プロセッサ１－ｉから送信されたフォーク要求７ａを廃棄する。従って、プロセッサ１－ｉで実行された今回のフォーク命令は無効化され、当該フォーク命令による子スレッドのフォークは断念される。

【００４３】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスに一致すると（ステップＳ２３でＹＥＳ）、フォークドビット２８がセットされていれば（ステップＳ２７でＹＥＳ）、アンドゲート３０の出力が論理“１”となり、実行ユニット２３に割り込みがかかり、当該プロセッサ１－ｉはスレッド終了通知７ｄをスレッド管理部３に送信し（ステップＳ２８）、スレッド管理部３からスレッド終了許可７ｅを受信した時点でスレッドの処理を終了する（ステップＳ２９）。しかし、フォークドビット２８がセットされていなければ、ＰＣ２１に従って命令の実行を継続して実行する（ステップＳ２２）。

【００４４】本実施例のマルチスレッド実行方法の実行シーケンスの一例を図８（ａ）に示す。この実行シーケンスは同図（ｂ）に示すような制御依存投機処理において投機が失敗した場合のシーケンスを示しており、プロセッサ＃０からプロセッサ＃１にフォークされた制御依存投機にかかる子スレッドは次の子スレッドのフォーク時にキャンセルされている。一方、図８（ｃ）、（ｄ）は同じような制御依存投機を従来のマルチスレッド実行方法で実行する際の実行シーケンスと制御依存投機処理を示しており、プロセッサ＃０からプロセッサ＃１にフォークした制御依存投機にかかる子スレッドをスレッド破棄命令（ａｂｏｒｔ）によってキャンセルしてから、次の子スレッドをフォークしている。本実施例では、２度目のフォークで既に存在する子スレッドをキャンセルするため、並列化プログラムへのスレッド破棄命令の挿入が不要である。また、子スレッドの開始点の直前へのターム命令（ｔｅｒｍ）の挿入が不要である。

【００４５】本実施例のマルチスレッド実行方法の実行シーケンスの別の例を図９（ａ）に示す。この図９（ａ）は、プロセッサ＃０で実行しているスレッドの最初のフォーク命令Ａの時点で、フォーク先プロセッサ＃１がビジー状態の場合を想定している。本実施例では、このような場合は当該フォーク命令Ａによるフォークが即断念される。このようにフォーク命令Ａのフォークを断念してもプログラムの処理は正しく遂行される。これを図１（ｅ）に示した並列化プログラムを例に説明すると、以下のようになる。図１（ｅ）中のフォーク命令ｆｏｒｋ　ｔｈ１がフォーク命令Ａに、フォーク命令ｆｏｒｋ　ｔｈ２がフォーク命令Ｂにそれぞれ対応するため、フォーク命令Ａが無効化され、フォーク命令Ｂがフォークされた場合の実行シーケンスは図９（ｂ）のようになる。プロセッサ＃０では命令０、命令１がその順

に、プロセッサ＃１では命令２、命令３、命令４がその順にそれぞれ実行されており、プログラムの処理は支障なく行える。

【００４６】同様に図９（ａ）においてフォーク命令Ｂの時点で、なおもフォーク先プロセッサ＃１がビジー状態であると、フォーク命令Ｂも無効化される。図１（ｅ）の並列化プログラムの場合、その実行シーケンスは図９（ｃ）に示すようになり、プロセッサ＃０において、命令０、命令１、命令２、命令３、命令４がこの順に逐次に実行されることになる。

【００４７】次に、本実施例のマルチスレッド実行方法で実行される並列化プログラムの生成方法について説明する。

【００４８】図１０を参照すると、コンパイラ４１は、逐次処理プログラム４２を入力し、制御及びデータフロー解析部４４によって逐次処理プログラム４２の制御フロー及びデータフローを解析して、基本ブロック或いは複数の基本ブロックを並列化の単位、すなわちスレッドに分割し、次いで並列化コード挿入部４５によって並列化のためのコードを挿入して、複数のスレッドに分割された並列化プログラム４３を生成して出力する。

【００４９】一般に並列化コードとしては、フォーク命令、ターム命令などがある。本実施例では並列プログラムの生成時に、フォーク点にフォーク命令が挿入される。しかし、子スレッドの開始点の直前に従来挿入されていたターム命令は挿入されない。また、図９（ａ）のフォーク命令Ａのように本実施例ではフォーク不可能ならばフォーク命令が無効化されるため、コンパイラ４１はフォーク命令が無効化されても逐次処理プログラム４２の正常動作が保証される並列化プログラム４３を生成する。一般に、並列化プログラム４３中からフォーク命令（それに対応するターム命令があればそのターム命令も）を全て取り除いた状態の制御フローが逐次処理プログラム４２の制御フローと等価であれば、逐次処理プログラム４２の動作を保証できる並列化プログラム４３となる。

【００５０】以上説明したように本実施例によれば、フォーク１回制限の保証のない並列化プログラムであっても実行時にフォーク１回制限を保証することができる。また、後述する第４の実施例のように親スレッドの最初のフォーク命令時点で隣接プロセッサがビジー状態の場合にフォーク命令をウエイトさせると処理が中断するが、本実施例ではそのような場合でも処理の中断無しにプログラムの処理を進めることができる。更に、後述する第５の実施例のようにフォーク不可能な場合にレジスタファイルの内容を退避バッファに退避して後刻におけるフォークを可能にする構成では、ハードウェア量が退避バッファの分だけ増加し、また退避バッファもオペレーティングシステムのプロセス切り替え時の退避、復元対象となるためにプロセス切り替えオーバーヘッドが増大するが、本実施例ではそのような問題も解消される。また後述する第６の実施例のように並列化プログラム中にターム命令を記述する必要がなく、プログラムサイズのコンパクト化による命令メモリの容量削減等が可能となる。

【００５１】

【第１の発明の第２の実施例】第１の実施例では、親スレッドのフォーク点でフォーク可能でなければフォークを即断念したが、本実施例では、親スレッドのレジスタファイルが更新される前にフォーク先プロセッサがフリー状態になるとフォークを行う。以下、第１の実施例との相違点を中心に本実施例を説明する。

【００５２】図１１を参照すると、本実施例における並列プロセッサシステムの各々のプロセッサ１－ｉは、図６に示した構成に加えて、フォーク有効ビット３３を含んで構成されている。フォーク有効ビット３３は、実行ユニット２３がフォーク命令を実行したときに出力するフォーク信号３４でセットされ、スレッド管理部３から受信されるフォーク応答７ｂ及び実行ユニット２３が親スレッドのレジスタファイル２５中の何れかのレジスタを更新したときに出力するレジスタ更新信号３５によってリセットされる。フォーク有効ビット３３の出力がスレッド管理部３に対するフォーク要求７ａとなり、フォーク有効ビット３３がセットされている間、フォーク要求７ａが送出し続けられる。

【００５３】前述の図４を参照すると、スレッド管理部３のスレッド管理シーケンサ１１は、或るクロックのタイミングでプロセッサ１－ｉからフォーク要求７ａを受信した際、子スレッド生成済信号が０且つ隣接プロセッサ１－ｉがビジー状態のとき（ステップＳ２でＮＯ）、当該フォーク要求７ａは破棄したが、本実施例では、プロセッサ１－ｉはフォーク有効ビット３３がセットされている間、フォーク要求７ａを送出し続けているので、次回のクロックのタイミングでスレッド管理部３がプロセッサ１－ｉからフォーク要求７ａを再び受信することになり、図４の処理が繰り返される。即ち、フォーク点でフォーク不可能な場合、フォーク命令は保留にされ、フォーク可能となった時点で実行されることになる。

【００５４】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図１２に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８及びフォーク有効ビット３３がリセットされ、またスレッドを実行中であればそれがキャンセルされる（ステップＳ２１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ２２）。

【００５５】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ２４でＹＥＳ）、実行ユニット２３はフォーク先アドレスをレジスタ２７に保

存し（ステップＳ２５）、フォーク信号３４によってフォーク有効ビット３３をセットすることにより（ステップＳ２６）、フォーク先アドレスとフォークドビット２８の値とを伴ったフォーク要求７ａをスレッド管理部３に送信する。また、実行ユニット２３はレジスタファイル２５中の何れかのレジスタを更新すると（ステップＳ３１でＹＥＳ）、レジスタ更新信号３５を出力してフォーク有効ビット３３をリセットする（ステップＳ３２）。従って、プロセッサ１－ｉからは、フォーク命令実行時点からレジスタファイル２５が最初に更新される迄の期間中、フォーク要求７ａがスレッド管理部３に送出し続けられる。

【００５６】スレッド管理部３は、プロセッサ１－ｉからフォーク要求７ａが送出されている期間内で、隣接プロセッサ１－ｊに対するフォークが可能になると、プロセッサ１－ｉにフォーク応答７ｂを送信すると同時に他プロセッサ１－ｊに対してスレッド開始要求７ｃを送出することで子スレッドをフォークする（ステップＳ４）。プロセッサ１－ｉに出されたフォーク応答７ｂによって、フォークドビット２８がセットされると共にフォーク有効ビット３３はリセットされ、またレジスタ転送ユニット２６によってレジスタファイル２５の内容がフォーク先プロセッサ１－ｊに転送される（ステップＳ３０）。なお、レジスタ転送ユニット２６によるレジスタファイル２５の転送中、実行ユニット２３からレジスタファイル２５への書き込みは待たされる。

【００５７】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスに一致したときの動作は第１の実施例と同様である。

【００５８】本実施例のマルチスレッド実行方法の実行シーケンスの一例を図１３に示す。フォーク命令Ａから下に延びる矢印は、プロセッサ＃０においてレジスタファイル２５が全く更新されていない期間を示す。この図１３は、プロセッサ＃０で実行しているスレッドの最初のフォーク命令Ａの時点で、フォーク先プロセッサ＃１がビジー状態であったが、プロセッサ＃０のレジスタファイル２５が全く更新されていない期間内にフォーク先プロセッサ＃１がフリー状態になった場合を想定している。本実施例では、このような場合はフォーク命令Ａによるフォークが行われる。

【００５９】以上説明したように本実施例によれば、第１の実施例と同様な効果が得られると共に、親スレッドの実行開始後、フォーク命令の時点でフォークできなくても、レジスタファイルが更新される前にフォーク先プロセッサがフリー状態になればフォークを行うため、第１の実施例に比べてフォークされる可能性が高まり、スレッド実行の並列度が向上する。

【００６０】

【第１の発明の第３の実施例】第２の実施例では、親スレッドのフォーク点でフォーク可能でなければフォークを一旦保留にし、親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他プロセッサが生じなかった場合に当該フォークを断念したが、本実施例では、親スレッドのレジスタファイルが更新されても、その更新が子スレッドに継承すべきレジスタでなければフォークを行う。以下、第２の実施例との相違点を中心に本実施例を説明する。

【００６１】図１４を参照すると、本実施例における並列プロセッサシステムの各々のプロセッサ１－ｉは、図１１に示した構成に加えて、レジスタファイル２５の各レジスタ２４－ｋ（ｋ＝０～ｍ）に１対１に対応し、対応するレジスタ２４－ｋが子スレッドへ継承すべきレジスタであるときに限りセットされるクリエイトビット３６－ｋと、各レジスタ２４－ｋに１対１に対応し、対応するレジスタ２４－ｋのクリエイトビット３６－ｋの出力と実行ユニット２３がレジスタ２４－ｋを更新したときに出力するレジスタ更新信号３７－ｋとを入力とするアンドゲート３８－ｋと、アンドゲート３８－ｋの出力の論理和信号であるフォーク無効信号４０を出力するオアゲート３９とを含んで構成されている。そして、図１４のレジスタ更新信号３５に代えて、フォーク無効信号４０がフォーク有効ビット３３にリセット信号として出力されている。また、各クリエイトビット３６－ｋの値がレジスタ転送ユニット２６に出力されており、レジスタ転送ユニット２６はレジスタファイル２５のレジスタ２４－ｋのうち、対応するクリエイトビット３６－ｋがセットされているレジスタのみをフォーク先プロセッサのレジスタファイルに転送するように構成されている。

【００６２】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図１５に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８及びフォーク有効ビット３３がリセットされ、またスレッドを実行中であればそれがキャンセルされる（ステップＳ２１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ２２）。

【００６３】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ２４でＹＥＳ）、実行ユニット２３はフォーク先アドレスをレジスタ２７に保存すると共に、全てのクリエイトビット３６－ｋのセットアップを行う（ステップＳ２５）。つまり、レジスタファイル２５のレジスタ２４－ｋのうち、子スレッドに継承すべきレジスタに対応するクリエイトビット３６－ｋはセットし、継承する必要のないレジスタに対応するクリエイトビット３６－ｋはリセットされたままにする。そして、フォーク信号３４によってフォーク有効ビット３３をセットすることにより（ステップＳ２６）、フォーク先アドレスとフォークドビット２８の値とを伴

ったフォーク要求７ａをスレッド管理部３に送信する。また、実行ユニット２３はレジスタファイル２５中のレジスタ２４－ｋを更新すると（ステップＳ３１でＹＥＳ）、その更新したレジスタ２４－ｋに対応するレジスタ更新信号３７－ｋを論理“１”とする。これにより、若し更新されたレジスタ２４－ｋが子スレッドへ継承すべきレジスタであった場合、そのレジスタ２４－ｋに対応するアンドゲート３８－ｋの出力が論理“１”となり、オアゲート３９からフォーク無効信号４０が出力されてフォーク有効ビット３３がリセットされる（ステップＳ３２）。つまり、プロセッサ１－ｉからは、フォーク命令実行時点からレジスタファイル２５中の子スレッドへの継承レジスタの何れかが最初に更新される迄の期間中、フォーク要求７ａがスレッド管理部３に送出し続けられる。

【００６４】スレッド管理部３は、プロセッサ１－ｉからフォーク要求７ａが送出されている期間内で、隣接プロセッサ１－ｊに対するフォークが可能になると、プロセッサ１－ｉにフォーク応答７ｂを送信すると同時に他プロセッサ１－ｊに対してスレッド開始要求７ｃを送出することで子スレッドをフォークする（ステップＳ４）。プロセッサ１－ｉに出されたフォーク応答７ｂによって、フォークドビット２８がセットされると共にフォーク有効ビット３３はリセットされ、またレジスタ転送ユニット２６によってレジスタファイル２５のレジスタの内、少なくとも子スレッドに継承すべきレジスタがフォーク先プロセッサ１－ｊに転送される（ステップＳ３０）。

【００６５】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスに一致したときの動作は第２の実施例と同様である。

【００６６】本実施例のマルチスレッド実行方法では、図１３中のフォーク命令Ａから下に述びる矢印は、プロセッサ＃０においてレジスタファイル２５のレジスタの内、子スレッドに継承すべきレジスタが全く更新されていない期間となる。従って、第１の実施例は勿論のこと、第２の実施例に比べてもフォークの可能性をより高めることができる。

【００６７】本実施例では、親スレッドのフォーク点で子スレッドへ継承すべきレジスタが判明している必要がある。このため、図１０に示したコンパイラ４１における制御及びデータフロー解析部４４では、フォークする子スレッド毎に、親スレッドから子スレッドへ継承すべきレジスタを調査し、並列化コード挿入部４５ではその調査結果に基づいて、子スレッドへ継承すべきレジスタを指定する記述を並列化プログラム４３に挿入する。子スレッドへ継承すべきレジスタの指定は、フォーク命令で指定する方法、フォーク命令とは別の専用の命令で指定する方法などが利用できる。

【００６８】なお、本実施例におけるレジスタ転送ユニット２６は、クリエイトビット３６－ｋを参照することにより、親スレッドのレジスタファイル２５のうち子スレッドに継承すべきレジスタだけをフォーク先プロセッサのレジスタファイルに転送するようにしたが、別の実施例として、レジスタファイル２５の先頭のレジスタから順に所定の順番でレジスタの転送を行うシーケンスを開始し、クリエイトビット３６－ｋがセットされているレジスタの全ての転送が完了した時点で転送シーケンスを停止するようにしても良い。この方法では、子スレッドに継承する必要のないレジスタも転送される場合があるが、転送シーケンスが簡素化される利点がある。勿論、別の実施例として、クリエイトビット３６－ｋを一切参照せずに常に全レジスタを転送するようにレジスタ転送ユニット２６が構成されていても良い。更に、子スレッドに継承すべきレジスタでも、フォーク先プロセッサの当該レジスタの値がフォーク時点で既に親スレッド側と同じ値になっている場合にはあえて転送する必要がない点に着目して、子スレッドに継承すべきレジスタのうち、親スレッド側と異なる値になっているレジスタを検出し、この検出したレジスタだけをレジスタ転送ユニット２６からフォーク先プロセッサに転送するようにしても良い。

【００６９】以上説明したように本実施例によれば、第２の実施例と同様な効果が得られると共に、親スレッドのレジスタファイルの更新があっても、その更新が子スレッドに継承すべきレジスタでなければフォークを行うため、第２の実施例に比べてフォークされる可能性をより高めることができ、従ってスレッド実行の並列度をより向上することができる。

【００７０】

【第１の発明の第４の実施例】本実施例は、親スレッドの実行開始後、フォーク命令の時点で隣接プロセッサがビジー状態であった場合、隣接プロセッサがフリー状態になるまでフォーク命令の実行をウエイトするようにした点で、第１乃至第３の実施例と相違する。以下、第２の実施例との相違点を中心に本実施例を説明する。

【００７１】図１６を参照すると、本実施例における並列プロセッサシステムの各々のプロセッサ１－ｉは、図１１に示した構成とほぼ同様な構成を有するが、スレッド管理部３から送信されるフォーク応答７ｂが実行ユニット２３にも入力されており、実行ユニット２３はフォーク命令の実行時、フォーク応答７ｂが返却されるまでフォーク命令の実行をウエイトする点で相違する。また、フォーク有効ビット３３を実行ユニット２３からリセットするレジスタ更新信号３５は存在しない。

【００７２】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図１７に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセ

ッサ１－ｉのフォークドビット２８及びフォーク有効ビット３３がリセットされ、またスレッドを実行中であればそれがキャンセルされる（ステップＳ２１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ２２）。

【００７３】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ２４でＹＥＳ）、実行ユニット２３はフォーク先アドレスをレジスタ２７に保存し（ステップＳ２５）、フォーク信号３４によってフォーク有効ビット３３をセットすることにより（ステップＳ２６）、フォーク先アドレスとフォークドビット２８の値とを伴ったフォーク要求７ａをスレッド管理部３に送信する。そして、スレッド管理部３からフォーク応答７ｂが返却されるのを待つ（ステップＳ４１）。

【００７４】スレッド管理部３は、プロセッサ１－ｉからフォーク要求７ａが送出されている期間内で、隣接プロセッサ１－ｊに対するフォークが可能になると、プロセッサ１－ｉにフォーク応答７ｂを送信すると同時に他プロセッサ１－ｊに対してスレッド開始要求７ｃを送出することで子スレッドをフォークする（ステップＳ４）。プロセッサ１－ｉは、スレッド管理部３からフォーク応答７ｂを受信すると、フォークドビット２８をセットすると共にフォーク有効ビット３３をリセットし、レジスタ転送ユニット２６によってレジスタファイル２５の内容をフォーク先プロセッサ１－ｊに転送する（ステップＳ４２）。

【００７５】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスに一致したときの動作は第２の実施例と同様である。

【００７６】

【第１の発明の第５の実施例】本実施例は、親スレッドの実行開始後、フォーク命令の時点で隣接プロセッサがビジー状態の場合、レジスタファイルの内容を退避させ、隣接プロセッサがフリー状態になった時点で前記退避した情報に基づいて子スレッドのフォークを行うようにした点で、第１乃至第４の実施例と相違する。以下、第４の実施例との相違点を中心に本実施例を説明する。

【００７７】図１８を参照すると、本実施例における並列プロセッサシステムの各々のプロセッサ１－ｉは、図１６に示した構成に加えて、退避バッファ４１と、退避バッファ有効ビット４２と、実行ユニット２３から出力される退避信号４３によって起動されるとレジスタファイル２５の内容を退避バッファ４１に退避する退避ユニット４４とを備え、レジスタ転送ユニット２６はレジスタファイル２５及び退避バッファ４２に接続されている。退避バッファ有効ビット４２は、実行ユニット２３から出力されるフォーク信号３４によってリセットされ、退避信号４３によってセットされ、また退避バッファ４１に基づくフォークが行われた場合にリセットされる。

【００７８】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図１９に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８及びフォーク有効ビット３３がリセットされ、またスレッドを実行中であればそれがキャンセルされる（ステップＳ２１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ２２）。

【００７９】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ２４でＹＥＳ）、実行ユニット２３はフォーク先アドレスをレジスタ２７に保存し（ステップＳ２５）、フォーク信号３４によってフォーク有効ビット３３をセットすることによりフォーク先アドレス及びフォークドビット２８の値を伴ったフォーク要求７ａをスレッド管理部３に送信する（ステップＳ２６）。このとき退避バッファ有効ビット４２がリセットされる。そして、スレッド管理部３から所定の時間内にフォーク応答７ｂが返却された場合（ステップＳ４３でＹＥＳ）、フォークドビット２８をセットすると共にフォーク有効ビット３３をリセットし、レジスタ転送ユニット２６によってレジスタファイル２５の内容をフォーク先プロセッサ１－ｊに転送する（ステップＳ４４）。

【００８０】他方、スレッド管理部３から所定の時間内にフォーク応答７ｂが返却されなかった場合（ステップＳ４３でＮＯ）、退避信号４３によって退避ユニット４４を起動することによりレジスタファイル２５の内容を退避バッファ４１へ退避させ、退避バッファ有効ビット４２をセットする（ステップＳ４５）。この退避バッファ有効ビット４２がセットされている間に、スレッド管理部３からフォーク応答７ｂを受信すると（ステップＳ４６、Ｓ４７でＹＥＳ）、レジスタ転送ユニット２６により退避バッファ４１に退避されているレジスタファイル２５の内容をフォーク先プロセッサ１－ｊに転送する（ステップＳ４８）。このとき、フォークドビット２８がセットされ、フォーク有効ビット３３及び退避バッファ有効ビット４２がリセットされる。なお、次のフォーク命令が実行される迄に前回のフォーク命令に対するフォーク応答７ｂがスレッド管理部３から返却されなかった場合、今回のフォーク命令の実行により退避バッファ有効ビットはリセットされるので（ステップＳ２６）、前回のフォーク命令は結果的に無効化される。

【００８１】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスに一致したときの動作は第４の実施例と同様である。

【００８２】

【第１の発明の第６の実施例】第１乃至第５の実施例で

は、各プロセッサ１－ｉはプログラムカウンタの値が有効な子スレッドの開始アドレスと一致したときにスレッドの処理を終了したが、本実施例では、各プロセッサは、各フォーク命令に対応して並列化プログラム中のフォーク先アドレスの直前に挿入されているターム命令のうち、有効な子スレッドをフォークしたフォーク命令に対応するターム命令によってスレッドの処理を終了する。以下、第１の実施例との相違点を中心に本実施例を説明する。

【００８３】図２０を参照すると、本実施例における並列プロセッサシステムの各々のプロセッサ１－ｉは、図６に示した構成のプロセッサにおける一致回路２９及びアンドゲート３０が省略され、フォークドビット２８の出力が実行ユニット２３に入力され、またスレッドＩＤを保持するレジスタ４５が設けられている。このレジスタ４５は、スレッド管理部３からのスレッド開始要求７ｃに付随するスレッドＩＤが初期設定され、プロセッサ１－ｉでフォーク命令が実行される毎に、そのフォーク命令で指定されたスレッドＩＤがセットされる。更に、スレッド管理部３へのフォーク要求７ａには、レジスタ２７に保存されたフォーク先アドレスとレジスタ４５に保存されたスレッドＩＤとフォークドビット２８の値とが付随する。

【００８４】スレッド管理部３は、図４のステップＳ４でフォーク先プロセッサ１－ｊへスレッド開始要求７ｃを送信する際、プロセッサ１－ｉからのフォーク要求７ａに付随するフォーク先アドレス及びスレッドＩＤを一緒に送信する。

【００８５】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図２１に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８がリセットされ、スレッド開始要求７ｃに付随するスレッドＩＤがレジスタ４５にセットされ、またスレッドを実行中であればそれがキャンセルされる（ステップＳ２１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ２２）。

【００８６】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ２４でＹＥＳ）、実行ユニット２３はフォーク先アドレスをレジスタ２７に保存すると共にそのフォーク命令で指定されたスレッドＩＤをレジスタ４５に保存（上書き）し（ステップＳ２５）、レジスタ２７に保存したフォーク先アドレス、レジスタ４５に保存したスレッドＩＤ及びフォークドビット２８の値を伴ったフォーク要求７ａをスレッド管理部３に送信する（ステップＳ２６）。スレッド管理部３は、隣接プロセッサ１－ｊに対してフォークが可能ならば、要求元のプロセッサ１－ｉに対してフォーク応答７ｂを返却し、隣接プロセッサ１－ｊに対してはフォーク先アドレス及びスレッドＩＤを付加したスレッド開始要求７ｃを送出する。フォーク応答７ｂを受信したプロセッサ１－ｉは、フォークドビット２８を１にセットし、レジスタ転送ユニット２６によって親スレッドのレジスタファイル２５の内容を通信バス６経由でフォーク先プロセッサ１－ｊのレジスタファイルに転送するレジスタ継承操作を行う（ステップＳ３０）。また、フォーク先プロセッサ１－ｊでは図２１のステップＳ２１以降の処理を実行する。

【００８７】他方、スレッド管理部３は、隣接プロセッサ１－ｊに対してフォークが不可能ならば、プロセッサ１－ｉから送信されたフォーク要求７ａを廃棄する。従って、プロセッサ１－ｉで実行された今回のフォーク命令は無効化され、当該フォーク命令による子スレッドのフォークは断念される。

【００８８】実行ユニット２３でデコードされた命令がターム命令の場合（ステップＳ５１でＹＥＳ）、実行ユニット２３は、フォークドビット２８が１にセットされており且つ当該ターム命令で指定されたスレッドＩＤがレジスタ４５に保存されているスレッドＩＤと一致した場合（ステップＳ５２、Ｓ５３でＹＥＳ）、当該ターム命令を実行することにより、スレッド終了通知７ｄを送出し（ステップＳ２８）、スレッド管理部３からスレッド終了許可７ｅを受信した時点でスレッドの処理を終了する（ステップＳ２９）。しかし、フォークドビット２８が１にセットされていないか、ターム命令で指定されたスレッドＩＤがレジスタ４５に保存されているスレッドＩＤと一致しない場合は、当該ターム命令を無効にし、ＰＣ２１に従って命令の実行を継続して実行する（ステップＳ２２）。

【００８９】本実施例では、各プロセッサはターム命令によってスレッドの処理を終了するため、図１０のコンパイラ４１における並列化コード挿入部４５は、子スレッドの開始点の直前に、当該子スレッドをフォークするフォーク命令に付加したスレッドＩＤと同じスレッドＩＤを持つターム命令を挿入する。

【００９０】本実施例と同様に、第２乃至第５の実施例において、各プロセッサが各フォーク命令に対応して並列化プログラム中のフォーク先アドレスの直前に挿入されているターム命令のうち、有効な子スレッドをフォークしたフォーク命令に対応するターム命令によってスレッドの処理を終了するように変形することができる。

【００９１】以上説明したように第１の発明によれば、フォーク１回制限の保証のない並列化プログラムであってもフォーク１回モデルによるマルチスレッド実行が可能になると共に以下のような効果が得られる。

（１）親スレッドのフォーク命令毎に親スレッドから生成された子スレッドが既に存在する場合にはその子スレッドをキャンセルすることで、プログラム実行時にフォーク１回制限を保証する為、動作が決定的であり、制御

が簡単である。
（２）後述する第２の発明に比べてスレッドの粒度が小さくなり、粒度を揃え易い。
（３）スレッド破棄命令（ａｂｏｒｔ）無しで、制御依存投機処理が可能になる。
【００９２】次に第２の発明の実施例について図面を参照して詳細に説明する。
【００９３】
【第２の発明の第１の実施例】本実施例にかかる並列プロセッサシステムは、図２に示した並列プロセッサシステムと同様な構成を有する。但し、各プロセッサ１－ｉにおける仮実行用バッファを用いたスレッド実行の取り消し（キャンセル）機能は必須でない。また、各プロセッサ１－ｉは、信号線２－ｉを通じてスレッド管理部３に対してフォーク要求７ａを送信する際、子スレッドのフォーク先アドレス（開始ＰＣ値）を付随させるが、子スレッド生成済み信号は付随させない。
【００９４】図２２を参照すると、本実施例の場合、スレッド管理部３のスレッド管理シーケンサ１１は、或るクロックのタイミングで何れかのプロセッサ１－ｉからフォーク要求ａを受信すると、隣接するプロセッサ１－ｊの状態をプロセッサ状態テーブル１２で調べ、フリー状態であれば（ステップＳ６１でＹＥＳ）、フォーク可能なため、プロセッサ状態テーブル１２における当該プロセッサ１－ｊに対応するエントリ１３－ｊをフリー状態からビジー状態に更新し（ステップＳ６２）、フォーク要求７ａに付随するフォーク先アドレスを添えたスレッド開始要求７ｃをフォーク先プロセッサ１－ｊに送信すると共に、要求元のプロセッサ１－ｉに対してフォーク応答７ｂを返却する（ステップＳ６３）。隣接するプロセッサ１－ｊがビジー状態であれば（ステップＳ６１でＮＯ）、フォーク不可能なので、スレッド管理シーケンサ１１は、当該フォーク要求７ａを廃棄する（ステップＳ６４）。スレッド管理シーケンサ１１が何れかのプロセッサ１－ｉからスレッド終了通知７ｄを受信したときの処理は図５と同じである。
【００９５】図２３を参照すると、各々のプロセッサ１－ｉは、スレッド管理部３から送信されたスレッド開始要求７ｃに付随する開始アドレス値がセットされ、その後に適宜歩進されるＰＣ２１と、ＰＣ２１に従ってメモリ５からスレッドの命令をフェッチする命令フェッチユニット２２と、フェッチされた命令をデコードし、実行する実行ユニット２３と、汎用レジスタ２４－０～２４－ｍの集合であるレジスタファイル２５と、フォーク先プロセッサに対して通信バス６経由でレジスタファイル２５の内容を転送するレジスタ転送ユニット２６と、フォーク命令実行時に実行ユニット２３からスレッド管理部３に送信されるフォーク要求７ａに付随するフォーク先アドレスを保存するレジスタ２７と、フォーク要求７ａに対するフォーク応答７ｂによってセットされるフォークドビット２８と、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスと一致するか否かを判定する一致回路２９と、フォークドビット２８及び一致回路２９の出力の論理積信号を実行ユニット２３に出力するアンドゲート３０とを含んで構成され、フォークドビット２８の値は実行ユニット２３にも入力されている。
【００９６】各々のプロセッサ１－ｉは、スレッド開始要求７ｃによって、それに付随する開始アドレスからスレッドの実行を開始する。また、実行ユニット２３は、フォーク命令のデコード時点で、フォークドビット２８がセットされているか否かを調べ、セットされていなければ当該フォーク命令を実行するが、セットされていれば当該フォーク命令を無効にする。さらに実行ユニット２３は、フォークドビット２８がセットされている状態においてＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスと一致することによりアンドゲート３０の出力が論理“１”になると、スレッドの処理を終了すべく、スレッド管理部３に対してスレッド終了通知７ｄを送信する。ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスと一致しても、フォークドビット２８がセットされていなければ、アンドゲート３０の出力は論理“１”にならないため、実行ユニット２３はＰＣ２１に従って命令の実行を継続する。
【００９７】レジスタ転送ユニット２６は、フォークドビット２８がセットされるタイミングでフォーク先プロセッサへのレジスタ転送を開始する。レジスタ転送ユニット２６は、例えば、通信バス６のバス幅によって一度に転送できる数のレジスタ毎に、レジスタファイル２５のレジスタの値とレジスタ番号（レジスタアドレス）とをフォーク先プロセッサのレジスタファイルへ送信し、受信側のレジスタファイル２５では該当するレジスタを書き換える。
【００９８】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図２４に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８がリセットされる（ステップＳ７１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ７２）。
【００９９】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ７４でＹＥＳ）、実行ユニット２３はフォークドビット２８が１にセットされていれば（ステップＳ７５でＹＥＳ）、つまり既に１回フォークを行っていれば、今回のフォーク命令を無効化し、次の命令を実行する（ステップＳ７２）。他方、フォークドビット２８が０であれば（ステップＳ７５でＮＯ）、つまり親スレッドの実行後、フォークを１回も行っていなければ、フォーク先アドレスをレジスタ２７に保存し（ステップＳ７６）、このレジスタ２７に保存し

たフォーク先アドレスを伴ったフォーク要求７ａをスレッド管理部３に送信する（ステップＳ７７）。

【０１００】スレッド管理部３は、図２２を参照して説明したように隣接プロセッサ１－ｊに対してフォークが可能ならば、要求元のプロセッサ１－ｉに対してフォーク応答７ｂを返却し、隣接プロセッサ１－ｊに対してはスレッド開始要求７ｃを送出する。フォーク応答７ｂを受信したプロセッサ１－ｉは、フォークドビット２８を１にセットし、レジスタ転送ユニット２６によって親スレッドのレジスタファイル２５の内容を通信バス６経由でフォーク先プロセッサ１－ｊのレジスタファイルに転送するレジスタ継承操作を行う（ステップＳ８１）。また、フォーク先プロセッサ１－ｊでは図２４のステップＳ７１以降の処理を実行する。

【０１０１】他方、スレッド管理部３は、隣接プロセッサ１－ｊに対してフォークが不可能ならば、プロセッサ１－ｉから送信されたフォーク要求７ａを廃棄する（ステップＳ６４）。従って、プロセッサ１－ｉで実行された今回のフォーク命令は無効化され、当該フォーク命令による子スレッドのフォークは断念される。

【０１０２】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスに一致すると（ステップＳ７３でＹＥＳ）、フォークドビット２８がセットされていれば（ステップＳ７８でＹＥＳ）、アンドゲート３０の出力が論理“１”となり、実行ユニット２３に割り込みがかかり、当該プロセッサ１－ｉはスレッド終了通知７ｄをスレッド管理部３に送信し（ステップＳ７９）、スレッド管理部３からスレッド終了許可７ｅを受信した時点でスレッドの処理を終了する（ステップＳ８０）。しかし、フォークドビット２８がセットされていなければ、ＰＣ２１に従って命令の実行を継続して実行する（ステップＳ７２）。

【０１０３】本実施例のマルチスレッド実行方法の実行シーケンスの例を図２５（ａ）に示す。この例は、プロセッサ＃０で実行している親スレッドの最初のフォーク命令Ａの時点で、フォーク先プロセッサ＃１がフリー状態の場合を想定している。この場合、親スレッド中のフォーク命令Ｂなどフォーク命令Ａ以外の全てのフォーク命令は無効化される。

【０１０４】本実施例のマルチスレッド実行方法の実行シーケンスの別の例を図２５（ｂ）に示す。この例は、プロセッサ＃０で実行しているスレッドの最初のフォーク命令Ａの時点で、フォーク先プロセッサ＃１がビジー状態の場合を想定しており、フォーク命令Ａは無効化されている。また、次のフォーク命令Ｂの時点でフォーク先プロセッサ＃１がフリー状態になっていた為、フォークが行われている例を示す。この場合、親スレッド中のフォーク命令Ｂ以降の全てのフォーク命令が無効化される。

【０１０５】図２５（ａ）の実行シーケンスを図１（ｅ）に示した並列化プログラムに当てはめると、図１（ｅ）中のフォーク命令ｆｏｒｋ　ｔｈ１がフォーク命令Ａに、フォーク命令ｆｏｒｋ　ｔｈ２がフォーク命令Ｂにそれぞれ対応する。このためフォーク命令Ａが実行され、フォーク命令Ｂが無効化された場合の実行シーケンスは図２５（ｃ）のようになる。プロセッサ＃０では命令０、命令１、命令２がその順に、プロセッサ＃１では命令３、命令４がその順にそれぞれ実行されており、プログラムの処理は支障なく行える。

【０１０６】また図２５（ｂ）の実行シーケンスを図１（ｅ）に示した並列化プログラムに当てはめると、図２５（ｄ）のようになる。プロセッサ＃０では命令０、命令１がその順に、プロセッサ＃１では命令２、命令３、命令４がその順にそれぞれ実行されており、プログラムの処理は支障なく行える。

【０１０７】なお、フォーク命令Ａ及びＢが無効化されると実行シーケンスは図２５（ｅ）に示すようになり、プロセッサ＃０において、命令０、命令１、命令２、命令３、命令４がこの順に逐次に実行されることになる。

【０１０８】本実施例のマルチスレッド実行方法で実行される並列化プログラムの生成方法は第１の発明の第１の実施例と同じである。

【０１０９】以上説明したように本実施例によれば、フォーク１回制限の保証のない並列化プログラムであっても実行時にフォーク１回制限を保証することができる。また、後述する第４の実施例のように親スレッドの最初のフォーク命令時点で隣接プロセッサがビジー状態の場合にフォーク命令をウエイトさせると処理が中断するが、本実施例ではそのような場合でも処理の中断無しにプログラムの処理を進めることができる。更に、後述する第５の実施例のようにフォーク不可能な場合にレジスタファイルの内容を退避バッファに退避して後刻におけるフォークを可能にする構成では、ハードウェア量が退避バッファの分だけ増加し、また退避バッファもオペレーティングシステムのプロセス切り替え時の退避、復元対象となるためにプロセス切り替えオーバーヘッドが増大するが、本実施例ではそのような問題も解消される。また後述する第６の実施例のように並列化プログラム中にターム命令を記述する必要がなく、プログラムサイズのコンパクト化による命令メモリの容量削減等が可能となる。

【０１１０】

【第２の発明の第２の実施例】第１の実施例では、親スレッドのフォーク点でフォーク可能でなければフォークを即断念したが、本実施例では、親スレッドのレジスタファイルが更新される前にフォーク先プロセッサがフリー状態になるとフォークを行う。以下、第１の実施例との相違点を中心に本実施例を説明する。

【０１１１】図２６を参照すると、本実施の形態における並列プロセッサシステムの各々のプロセッサ１－ｉ

は、図２３に示した構成に加えて、フォーク有効ビット３３を含んで構成されている。フォーク有効ビット３３は、実行ユニット２３がフォーク命令を実行したときに出力するフォーク信号３４でセットされ、スレッド管理部３から受信されるフォーク応答７ｂ及び実行ユニット２３が親スレッドのレジスタファイル２５中の何れかのレジスタを更新したときに出力するレジスタ更新信号３５によってリセットされる。フォーク有効ビット３３の出力がスレッド管理部３に対するフォーク要求７ａとなり、フォーク有効ビット３３がセットされている間、フォーク要求７ａが送出し続けられる。

【０１１２】前述の図２２を参照すると、スレッド管理部３のスレッド管理シーケンサ１１は、或るクロックのタイミングでプロセッサ１－ｉからフォーク要求７ａを受信した際、隣接プロセッサ１－ｉがビジー状態のとき、当該フォーク要求７ａは破棄したが（ステップＳ６４）、本実施例では、プロセッサ１－ｉはフォーク有効ビット３３がセットされている間、フォーク要求７ａを送出し続けているので、次回のクロックのタイミングでスレッド管理部３がプロセッサ１－ｉからフォーク要求７ａを再び受信することになり、図２２の処理が繰り返される。即ち、フォーク点でフォーク不可能な場合、フォーク命令は保留にされ、フォーク可能となった時点で実行されることになる。

【０１１３】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図２７に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８及びフォーク有効ビット３３がリセットされる（ステップＳ７１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ７２）。

【０１１４】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ７４でＹＥＳ）、実行ユニット２３はフォークドビット２８が１にセットされていれば（ステップＳ７５でＹＥＳ）、今回のフォーク命令を無効化し、次の命令を実行する（ステップＳ７２）。他方、フォークドビット２８が０であれば（ステップＳ７５でＮＯ）、フォーク先アドレスをレジスタ２７に保存し（ステップＳ７６）、フォーク有効ビット３３をセットすることにより、レジスタ２７に保存したフォーク先アドレスを伴ったフォーク要求７ａをスレッド管理部３に送信する（ステップＳ７７）。また、実行ユニット２３はレジスタファイル２５中の何れかのレジスタを更新すると（ステップＳ８２でＹＥＳ）、レジスタ更新信号３５を出力してフォーク有効ビット３３をリセットする（ステップＳ８３）。従って、プロセッサ１－ｉからは、フォーク命令実行時点からレジスタファイル２５が最初に更新される迄の期間中、フォーク要求７ａがスレッド管理部３に送出し続けられる。

【０１１５】スレッド管理部３は、プロセッサ１－ｉからフォーク要求７ａが送出されている期間内で、隣接プロセッサ１－ｊに対するフォークが可能になると、プロセッサ１－ｉにフォーク応答７ｂを送信すると同時に他プロセッサ１－ｊに対してスレッド開始要求７ｃを送出することで子スレッドをフォークする（ステップＳ６３）。プロセッサ１－ｉに出されたフォーク応答７ｂによって、フォークドビット２８がセットされると共にフォーク有効ビット３３はリセットされ、またレジスタ転送ユニット２６によってレジスタファイル２５の内容がフォーク先プロセッサ１－ｊに転送される（ステップＳ８１）。なお、レジスタ転送ユニット２６によるレジスタファイル２５の転送中、実行ユニット２３からのレジスタファイル２６への書き込みが待たされる。

【０１１６】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスに一致したときの動作は第１の実施例と同様である。

【０１１７】本実施例のマルチスレッド実行方法の実行シーケンスの一例を図２８に示す。フォーク命令Ａから下に延びる矢印は、プロセッサ＃０においてレジスタファイル２５が全く更新されていない期間を示す。この図２８は、プロセッサ＃０で実行しているスレッドの最初のフォーク命令Ａの時点で、フォーク先プロセッサ＃１がビジー状態であったが、プロセッサ＃０のレジスタファイル２５が全く更新されていない期間内にフォーク先プロセッサ＃１がフリー状態になった場合を想定している。本実施例では、このような場合はフォーク命令Ａによるフォークが行われる。

【０１１８】以上説明したように本実施例によれば、第１の実施例と同様な効果が得られると共に、親スレッドの実行開始後、フォーク命令の時点でフォークできなくても、レジスタファイルが更新される前に隣接プロセッサがフリー状態になればフォークを行うため、第１の実施例に比べてフォークされる可能性が高まり、スレッド実行の並列度が向上する。

【０１１９】

【第２の発明の第３の実施例】第２の実施例では、親スレッドのフォーク点でフォーク可能でなければフォークを一旦保留にし、親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他プロセッサが生じなかった場合に当該フォークを断念したが、本実施例では、親スレッドのレジスタファイルが更新されても、その更新が子スレッドに継承すべきレジスタでなければフォークを行う。以下、第２の実施例との相違点を中心に本実施例を説明する。

【０１２０】図２９を参照すると、本実施例における並列プロセッサシステムの各々のプロセッサ１－ｉは、図２６に示した構成に加えて、レジスタファイル２５の各レジスタ２４－ｋ（ｋ＝０～ｍ）に１対１に対応し、対

応するレジスタ２４－ｋが子スレッドへ継承すべきレジスタであるときに限りセットされるクリエイトビット３６－ｋと、各レジスタ２４－ｋに１対１に対応し、対応するレジスタ２４－ｋのクリエイトビット３６－ｋの出力と実行ユニット２３がレジスタ２４－ｋを更新したときに出力するレジスタ更新信号３７－ｋとを入力とするアンドゲート３８－ｋと、アンドゲート３８－ｋの出力の論理和信号であるフォーク無効信号４０を出力するオアゲート３９とを含んで構成されている。そして、図２６のレジスタ更新信号３５に代えて、フォーク無効信号４０がフォーク有効ビット３３にリセット信号として出力されている。また、各クリエイトビット３６－ｋの値がレジスタ転送ユニット２６に出力されており、レジスタ転送ユニット２６はレジスタファイル２５のレジスタ２４－ｋのうち、対応するクリエイトビット３６－ｋがセットされているレジスタのみをフォーク先プロセッサのレジスタファイルに転送するように構成されている。

【０１２１】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図３０に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８及びフォーク有効ビット３３がリセットされる（ステップＳ７１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ７２）。

【０１２２】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ７４でＹＥＳ）、実行ユニット２３はフォークドビット２８が１にセットされていれば（ステップＳ７５でＹＥＳ）、今回のフォーク命令を無効化し、次の命令を実行する（ステップＳ７２）。他方、フォークドビット２８が０であれば（ステップＳ７５でＮＯ）、実行ユニット２３はフォーク先アドレスをレジスタ２７に保存すると共に、全てのクリエイトビット３６－ｋのセットアップを行う（ステップＳ７６）。つまり、レジスタファイル２５のレジスタ２４－ｋのうち、子スレッドに継承すべきレジスタに対応するクリエイトビット３６－ｋはセットし、継承する必要のないレジスタに対応するクリエイトビット３６－ｋはリセットされたままにする。そして、フォーク信号３４によってフォーク有効ビット３３をセットすることにより（ステップＳ７７）、フォーク先アドレスを伴ったフォーク要求７ａをスレッド管理部３に送信する。また、実行ユニット２３はレジスタファイル２５中のレジスタ２４－ｋを更新すると（ステップＳ９１でＹＥＳ）、その更新したレジスタ２４－ｋに対応するレジスタ更新信号３７－ｋを論理“１”とする（ステップＳ９２）。これにより、若し更新されたレジスタ２４－ｋが子スレッドへ継承すべきレジスタであった場合、そのレジスタ２４－ｋに対応するアンドゲート３８－ｋの出力が論理“１”となり、オアゲート３９からフォーク無効信号４０が出力されてフォーク有効ビット３３がリセットされる。つまり、プロセッサ１－ｉからは、フォーク命令実行時点からレジスタファイル２５中の子スレッドへの継承レジスタの何れかが最初に更新される迄の期間中、フォーク要求７ａがスレッド管理部３に送出し続けられる。

【０１２３】スレッド管理部３は、プロセッサ１－ｉからフォーク要求７ａが送出されている期間内で、隣接プロセッサ１－ｊに対するフォークが可能になると、プロセッサ１－ｉにフォーク応答７ｂを送信すると同時に他プロセッサ１－ｊに対してスレッド開始要求７ｃを送出することで子スレッドをフォークする（ステップＳ６３）。プロセッサ１－ｉに出されたフォーク応答７ｂによって、フォークドビット２８がセットされると共にフォーク有効ビット３３はリセットされ、またレジスタ転送ユニット２６によってレジスタファイル２５のレジスタの内、少なくとも子スレッドに継承すべきレジスタがフォーク先プロセッサ１－ｊに転送される（ステップＳ８１）。

【０１２４】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスに一致したときの動作は第２の実施例と同様である。

【０１２５】本実施例のマルチスレッド実行方法では、図２８中のフォーク命令Ａから下に述びる矢印は、プロセッサ＃０においてレジスタファイル２５のレジスタの内、子スレッドに継承すべきレジスタが全く更新されていない期間となる。従って、第１の実施例は勿論のこと、第２の実施例に比べてもフォークの可能性をより高めることができる。

【０１２６】本実施例では、親スレッドのフォーク点で子スレッドへ継承すべきレジスタが判明している必要がある。このため、図１０に示したコンパイラ４１における制御及びデータフロー解析部４４では、フォークする子スレッド毎に、親スレッドから子スレッドへ継承すべきレジスタを調査し、並列化コード挿入部４５ではその調査結果に基づいて、子スレッドへ継承すべきレジスタを指定する記述を並列化プログラム４３に挿入する。子スレッドへ継承すべきレジスタの指定は、フォーク命令で指定する方法、フォーク命令とは別の専用の命令で指定する方法などが利用できる。

【０１２７】なお、本実施例におけるレジスタ転送ユニット２６は、クリエイトビット３６－ｋを参照することにより、親スレッドのレジスタファイル２５のうち子スレッドに継承すべきレジスタだけをフォーク先プロセッサのレジスタファイルに転送するようにしたが、別の実施例として、レジスタファイル２５の先頭のレジスタから順に所定の順番でレジスタの転送を行うシーケンスを開始し、クリエイトビット３６－ｋがセットされているレジスタの全ての転送が完了した時点で転送シーケンス

を停止するようにしても良い。この方法では、子スレッドに継承する必要のないレジスタも転送される場合があるが、転送シーケンスが簡素化される利点がある。勿論、別の実施例として、クリエイトビット３６－ｋを一切参照せずに常に全レジスタを転送するようにレジスタ転送ユニット２６が構成されていても良い。更に、子スレッドに継承すべきレジスタでも、フォーク先プロセッサの当該レジスタの値がフォーク時点で既に親スレッド側と同じ値に同じ値になっている場合にはあえて転送する必要がない点に着目して、子スレッドに継承すべきレジスタのうち、親スレッド側と異なる値になっているレジスタを検出し、この検出したレジスタだけをレジスタ転送ユニット２６からフォーク先プロセッサに転送するようにしても良い。

【０１２８】以上説明したように本実施例によれば、第２の実施例と同様な効果が得られると共に、親スレッドのレジスタファイルの更新があっても、その更新が子スレッドに継承すべきレジスタでなければフォークを行うため、第２の実施例に比べてフォークされる可能性をより高めることができ、従ってスレッド実行の並列度をより向上することができる。

【０１２９】

【第２の発明の第４の実施例】本実施例は、親スレッドの実行開始後、フォーク命令の時点で隣接プロセッサがビジー状態であった場合、隣接プロセッサがフリー状態になるまでフォーク命令の実行をウエイトするようにした点で、第１乃至第３の実施例と相違する。以下、第２の実施例との相違点を中心に本実施例を説明する。

【０１３０】図３１を参照すると、本実施例における並列プロセッサシステムの各々のプロセッサ１－ｉは、図２６に示した構成とほぼ同様な構成を有するが、スレッド管理部３から送信されるフォーク応答７ｂが実行ユニット２３にも入力されており、実行ユニット２３はフォーク命令の実行時、フォーク応答７ｂが返却されるまでフォーク命令の実行をウエイトする点で相違する。また、フォーク有効ビット３３を実行ユニット２３からリセットするレジスタ更新信号３５は存在しない。

【０１３１】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図３２に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８及びフォーク有効ビット３３がリセットされる（ステップＳ７１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ７２）。

【０１３２】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ７４でＹＥＳ）、実行ユニット２３はフォークドビット２８が１にセットされていれば（ステップＳ７５でＹＥＳ）、今回のフォーク命令を無効化し、次の命令を実行する（ステップＳ７２）。他方、フォークドビット２８が０であれば（ステップＳ７５でＮＯ）、フォーク先アドレスをレジスタ２７に保存し（ステップＳ７６）、フォーク有効ビット３３をセットすることにより、レジスタ２７に保存したフォーク先アドレスを伴ったフォーク要求７ａをスレッド管理部３に送信する（ステップＳ７７）。そして、スレッド管理部３からフォーク応答７ｂが返却されるのを待つ（ステップＳ１０１）。

【０１３３】スレッド管理部３は、プロセッサ１－ｉからフォーク要求７ａが送出されている期間内で、隣接プロセッサ１－ｊに対するフォークが可能になると、プロセッサ１－ｉにフォーク応答７ｂを送信すると同時に他プロセッサ１－ｊに対してスレッド開始要求７ｃを送出することで子スレッドをフォークする（ステップＳ６３）。プロセッサ１－ｉは、スレッド管理部３からフォーク応答７ｂを受信すると、フォークドビット２８をセットすると共にフォーク有効ビット３３をリセットし、レジスタ転送ユニット２６によってレジスタファイル２５の内容をフォーク先プロセッサ１－ｊに転送する（ステップＳ１０２）。

【０１３４】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスに一致したときの動作は第２の実施例と同様である。

【０１３５】

【第２の発明の第５の実施例】本実施例は、親スレッドの実行開始後、フォーク命令の時点で隣接プロセッサがビジー状態の場合、レジスタファイルの内容を退避させ、隣接プロセッサがフリー状態になった時点で前記退避した情報に基づいて子スレッドのフォークを行うようにした点で、第１乃至第４の実施例と相違する。以下、第４の実施例との相違点を中心に本実施例を説明する。

【０１３６】図３３を参照すると、本実施例における並列プロセッサシステムの各々のプロセッサ１－ｉは、図３１に示した構成に加えて、退避バッファ４１と、退避バッファ有効ビット４２と、実行ユニット２３から出力される退避信号４３によって起動されるとレジスタファイル２５の内容を退避バッファ４１に退避する退避ユニット４４とを備え、レジスタ転送ユニット２６はレジスタファイル２５及び退避バッファ４２に接続されている。退避バッファ有効ビット４２は、実行ユニット２３から出力されるフォーク信号３４によってリセットされ、退避信号４３によってセットされ、また退避バッファ４１に基づくフォークが行われた場合にリセットされる。

【０１３７】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図３４に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８及びフォーク有効ビ

ット３３がリセットされる（ステップＳ７１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ７２）。

【０１３８】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ７４でＹＥＳ）、実行ユニット２３はフォークドビット２８が１にセットされていれば（ステップＳ７５でＹＥＳ）、今回のフォーク命令を無効化し、次の命令を実行する（ステップＳ７２）。他方、フォークドビット２８が０であれば（ステップＳ７５でＮＯ）、フォーク先アドレスをレジスタ２７に保存し（ステップＳ７６）、フォーク有効ビット３３をセットすることにより、レジスタ２７に保存したフォーク先アドレスを伴ったフォーク要求７ａをスレッド管理部３に送信する（ステップＳ７７）。このとき退避バッファ有効ビット４２がリセットされる。そして、スレッド管理部３から所定の時間内にフォーク応答７ｂが返却された場合（ステップＳ１１１でＹＥＳ）、フォークドビット２８をセットすると共にフォーク有効ビット３３をリセットし、レジスタ転送ユニット２６によってレジスタファイル２５の内容をフォーク先プロセッサ１－ｊに転送する（ステップＳ１１２）。

【０１３９】他方、スレッド管理部３から所定の時間内にフォーク応答７ｂが返却されなかった場合（ステップＳ１１１でＮＯ）、退避信号４３によって退避ユニット４４を起動することによりレジスタファイル２５の内容を退避バッファ４１へ退避させ、退避バッファ有効ビット４２をセットする（ステップＳ１１３）。この退避バッファ有効ビット４２がセットされている間に、スレッド管理部３からフォーク応答７ｂを受信すると（ステップＳ１１４、Ｓ１１５でＹＥＳ）、レジスタ転送ユニット２６により退避バッファ４１に退避されているレジスタファイル２５の内容をフォーク先プロセッサ１－ｊに転送する（ステップＳ１１６）。このとき、フォークドビット２８がセットされ、フォーク有効ビット３３及び退避バッファ有効ビット４２がリセットされる。なお、次のフォーク命令が実行される迄に前回のフォーク命令に対するフォーク応答７ｂがスレッド管理部３から返却されなかった場合、今回のフォーク命令の実行により退避バッファ有効ビット４２はリセットされるので（ステップＳ７６）、前回のフォーク命令は結果的に無効化される。

【０１４０】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２７に保存されたフォーク先アドレスに一致したときの動作は第４の実施例と同様である。

【０１４１】

【第２の発明の第６の実施例】第１乃至第５の実施例では、各プロセッサ１－ｉはプログラムカウンタの値が有効な子スレッドの開始アドレスと一致したときにスレッドの処理を終了したが、本実施例では、各プロセッサは、各フォーク命令に対応して並列化プログラム中のフォーク先アドレスの直前に挿入されているターム命令のうち、有効な子スレッドをフォークしたフォーク命令に対応するターム命令によってスレッドの処理を終了する。以下、第１の実施例との相違点を中心に本実施例を説明する。

【０１４２】図３５を参照すると、本実施例における並列プロセッサシステムの各々のプロセッサ１－ｉは、図２３に示した構成のプロセッサにおける一致回路２９及びアンドゲート３０が省略され、スレッドＩＤを保持するレジスタ４５が設けられている。このレジスタ４５は、スレッド管理部３からのスレッド開始要求７ｃに付随するスレッドＩＤが初期設定され、プロセッサ１－ｉでフォーク命令が実行される毎に、そのフォーク命令で指定されたスレッドＩＤがセットされる。更に、スレッド管理部３へのフォーク要求７ａには、レジスタ２７に保存されたフォーク先アドレスとレジスタ４５に保存されたスレッドＩＤとが付随する。

【０１４３】スレッド管理部３は、図２２のステップＳ６３でフォーク先プロセッサ１－ｊへスレッド開始要求７ｃを送信する際、プロセッサ１－ｉからのフォーク要求７ａに付随するフォーク先アドレス及びスレッドＩＤを一緒に送信する。

【０１４４】スレッドの開始から終了までのプロセッサ１－ｉの処理の概要を図３６に示す。スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２８がリセットされ、スレッド開始要求７ｃに付随するスレッドＩＤがレジスタ４５にセットされる（ステップＳ７１）。以後、スレッドの命令のフェッチ、デコード、実行が継続して実行される（ステップＳ７２）。

【０１４５】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ７４でＹＥＳ）、実行ユニット２３はフォークドビット２８が１にセットされていれば（ステップＳ７５でＹＥＳ）、今回のフォーク命令を無効化し、次の命令を実行する（ステップＳ７２）。他方、フォークドビット２８が０であれば（ステップＳ７５でＮＯ）、実行ユニット２３はフォーク先アドレスをレジスタ２７に保存すると共にそのフォーク命令で指定されたスレッドＩＤをレジスタ４５に保存（上書き）し（ステップＳ７６）、レジスタ２７に保存したフォーク先アドレス及びレジスタ４５に保存したスレッドＩＤを伴ったフォーク要求７ａをスレッド管理部３に送信する（ステップＳ７７）。スレッド管理部３は、隣接プロセッサ１－ｊに対してフォークが可能ならば、要求元のプロセッサ１－ｉに対してフォーク応答７ｂを返却し、隣接プロセッサ１－ｊに対してはフォーク先アドレス及びスレッドＩＤを付加したスレッド開始要求７ｃを送出する。フォーク応答７ｂを受信したプロセッサ１

－ｉは、フォークドビット２８を１にセットし、レジスタ転送ユニット２６によって親スレッドのレジスタファイル２５の内容を通信バス６経由でフォーク先プロセッサ１－ｊのレジスタファイルに転送するレジスタ継承操作を行う（ステップＳ８１）。また、フォーク先プロセッサ１－ｊでは図３６のステップＳ７１以降の処理を実行する。

【０１４６】他方、スレッド管理部３は、隣接プロセッサ１－ｊに対してフォークが不可能ならば、プロセッサ１－ｉから送信されたフォーク要求７ａを廃棄する。従って、プロセッサ１－ｉで実行された今回のフォーク命令は無効化され、当該フォーク命令による子スレッドのフォークは断念される。

【０１４７】実行ユニット２３でデコードされた命令がターム命令の場合（ステップＳ１２１でＹＥＳ）、実行ユニット２３は、フォークドビット２８が１にセットされており且つ当該ターム命令で指定されたスレッドＩＤがレジスタ４５に保存されているスレッドＩＤと一致した場合（ステップＳ１２２、Ｓ１２３でＹＥＳ）、当該ターム命令を実行することにより、スレッド終了通知７ｄを送出し（ステップＳ７９）、スレッド管理部３からスレッド終了許可７ｅを受信した時点でスレッドの処理を終了する（ステップＳ８０）。しかし、フォークドビット２８が１にセットされていないか、ターム命令で指定されたスレッドＩＤがレジスタ４５に保存されているスレッドＩＤと一致しない場合は、当該ターム命令を無効にし、ＰＣ２１に従って命令の実行を継続して実行する（ステップＳ７２）。

【０１４８】本実施例では、各プロセッサはターム命令によってスレッドの処理を終了するため、図１０のコンパイラ４１における並列化コード挿入部４５は、子スレッドの開始点の直前に、当該子スレッドをフォークするフォーク命令に付加したスレッドＩＤと同じスレッドＩＤを持つターム命令を挿入する。

【０１４９】本実施例と同様に、第２乃至第５の実施例において、各プロセッサが各フォーク命令に対応して並列化プログラム中のフォーク先アドレスの直前に挿入されているターム命令のうち、有効な子スレッドをフォークしたフォーク命令に対応するターム命令によってスレッドの処理を終了するように変形することができる。

【０１５０】以上説明したように第２の発明によれば、フォーク１回制限の保証のない並列化プログラムであってもフォーク１回モデルによるマルチスレッド実行が可能になると共に、親スレッドの実行を開始したプロセッサで最初に子スレッドのフォークに成功したフォーク命令以外の全てのフォーク命令を無効化することで、プログラム実行時にフォーク１回制限を保証する為、動作が決定的であり、制御が簡単である利点がある。

【０１５１】以上、本発明を幾つかの実施例を挙げて説明したが、本発明は以上の実施例にのみ限定されず、その他各種の付加変更が可能である。例えば、前記各実施例では、複数のプロセッサに共通にスレッド管理部３を設ける集中スレッド管理型の並列プロセッサシステムに本発明を適用したが、文献１等に記載されるように各プロセッサ毎にスレッド管理部を設ける分散スレッド管理型の並列プロセッサシステムにも本発明は適用可能である。また、プロセッサ相互間を通信バス６によって接続したが、リング型フォークモデルにあっては隣接するプロセッサ間どうしをリング上に通信線で接続する形態の並列プロセッサシステムに対しても本発明は適用可能である。

【０１５２】

【発明の効果】以上説明したように本発明によれば、フォーク１回制限の保証のない並列化プログラムであってもプログラム実行時にフォーク１回制限を保証できるマルチスレッド実行方法及び並列プロセッサシステムが得られ、コンパイル段階でのフォーク１回制限を取り除くことができる。

【０１５３】また、第１及び第２の発明ともプログラム実行時にフォーク１回制限を保証する動作が決定的であり、制御が簡単である効果がある。

【０１５４】また、第２の発明によれば、スレッド破棄命令（ａｂｏｒｔ）無しで、制御依存投機処理が可能になる効果がある。

【０１５５】また、第１及び第２の発明における第１乃至第３の実施例によれば、退避バッファを持つことによるハードウェア量の増加、ＯＳのプロセス切り替え時におけるオーバヘッドの増大を防止しつつ、親スレッドのフォーク命令時点で子スレッドを生成できる空きのプロセッサが存在しない場合でも処理の中断無しにプログラムの処理を支障なく遂行することができる効果がある。

【０１５６】また、第１及び第２の発明における第２の実施例によれば、フォーク命令の時点でフォークできなくても、レジスタファイルが更新される前に子スレッドの実行を開始できる空きのプロセッサが生じるとフォークが可能になるため、第１の実施例に比べてフォークできる確率が高まり、スレッド実行の並列度を向上することができる。

【０１５７】また、第１及び第２の発明における第３の実施例によれば、親スレッドのレジスタファイルの更新があっても、その更新が子スレッドに継承すべきレジスタでなければフォークを行うため、第２の実施例に比べてフォークできる確率を高めることができ、スレッド実行の並列度をより一層向上することができる。

【０１５８】また、第１及び第２の発明における第１乃至第５の実施例によれば、子スレッドの開始点の直前にターム命令を置く必要がなくなり、ターム命令の削減によってプログラムサイズをコンパクト化でき、命令メモリに必要な容量の削減、命令フェッチ数の削減による処理性能の向上が可能となる。

【０１５９】また、第１及び第２の発明における第６の実施例によれば、従来と同様に子スレッドの開始点の直前にターム命令が置かれたプログラムを支障なく実行することが可能となる。

【図面の簡単な説明】

【図１】本発明の作用の説明図である。

【図２】本発明の並列プロセッサシステムの一例を示すブロック図である。

【図３】本発明の並列プロセッサシステムにおけるスレッド管理部の構成例を示すブロック図である。

【図４】本発明の並列プロセッサシステムにおけるスレッド管理部のスレッド管理シーケンサがプロセッサからフォーク要求を受信した際の処理例を示すフローチャートである。

【図５】本発明の並列プロセッサシステムにおけるスレッド管理部のスレッド管理シーケンサがプロセッサからスレッド終了通知を受信した際の処理例を示すフローチャートである。

【図６】本発明の第１の発明の第１の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図７】本発明の第１の発明の第１の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図８】本発明の第１の発明の第１の実施例にかかるマルチスレッド実行方法の実行シーケンスの一例を示す図である。

【図９】本発明の第１の発明の第１の実施例にかかるマルチスレッド実行方法の実行シーケンスの別の例を示す図である。

【図１０】本発明のマルチスレッド実行方法向けの並列化プログラムを生成するコンパイラの構成例を示すブロック図である。

【図１１】本発明の第１の発明の第２の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図１２】本発明の第１の発明の第２の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図１３】本発明の第１の発明の第２の実施例にかかるマルチスレッド実行方法の実行シーケンスの一例を示す図である。

【図１４】本発明の第１の発明の第３の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図１５】本発明の第１の発明の第３の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図１６】本発明の第１の発明の第４の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図１７】本発明の第１の発明の第４の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図１８】本発明の第１の発明の第５の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図１９】本発明の第１の発明の第５の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図２０】本発明の第１の発明の第６の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図２１】本発明の第１の発明の第６の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図２２】本発明の並列プロセッサシステムにおけるスレッド管理部のスレッド管理シーケンサがプロセッサからフォーク要求を受信した際の別の実施例を示すフローチャートである。

【図２３】本発明の第２の発明の第１の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図２４】本発明の第２の発明の第１の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図２５】本発明の第２の発明の第１の実施例にかかるマルチスレッド実行方法の実行シーケンスの一例を示す図である。

【図２６】本発明の第２の発明の第２の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図２７】本発明の第２の発明の第２の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図２８】本発明の第２の発明の第２の実施例にかかるマルチスレッド実行方法の実行シーケンスの一例を示す図である。

【図２９】本発明の第２の発明の第３の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図３０】本発明の第２の発明の第３の実施例にかかる

並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図３１】本発明の第２の発明の第４の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図３２】本発明の第２の発明の第４の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図３３】本発明の第２の発明の第５の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図３４】本発明の第２の発明の第５の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

【図３５】本発明の第２の発明の第６の実施例にかかる並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図３６】本発明の第２の発明の第６の実施例にかかる並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサの処理の一例を示すフローチャートである。

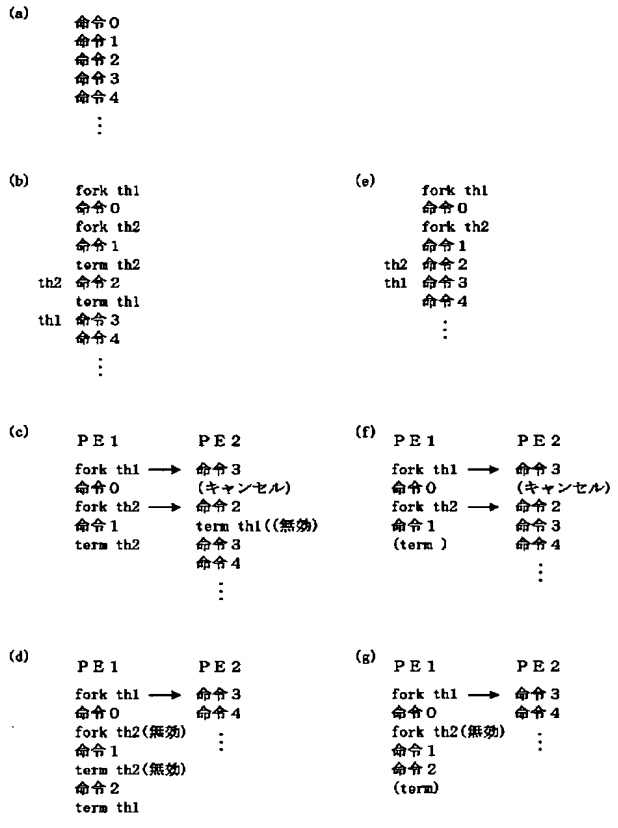
【図３７】従来のマルチスレッド実行方法の処理の概要を示す図である。

【図３８】従来の問題点の説明図である。

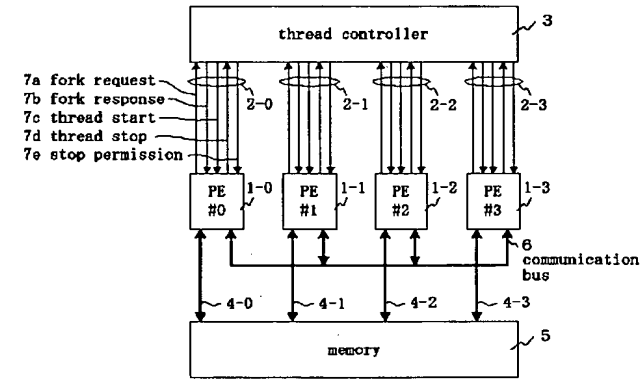
【符号の説明】

１－０～１－３…プロセッサ
２－０～２－３…信号線
３…スレッド管理部
４－０～４－３…信号線
５…メモリ
６…通信バス

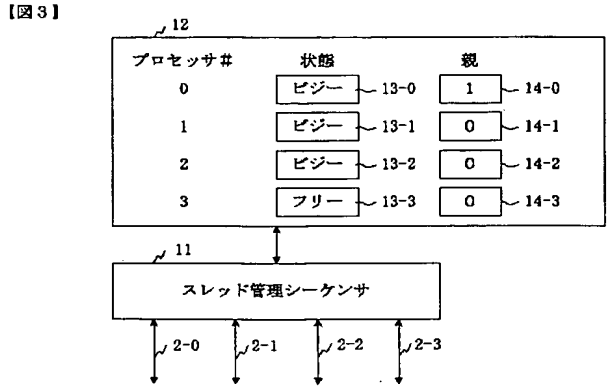
【図１】



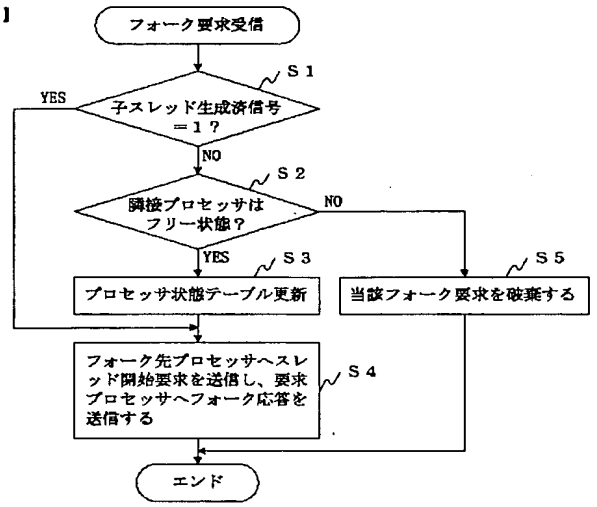
【図２】



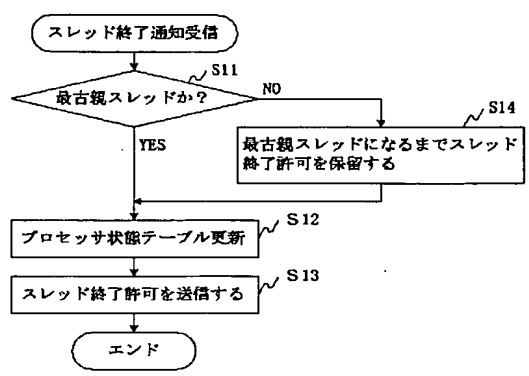
【図３】



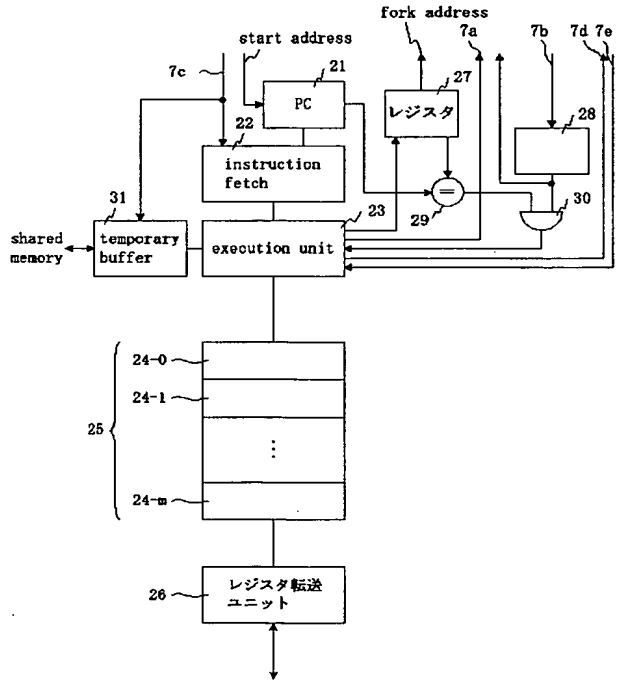
【図４】


【図４】
フォーク要求受信
S１
子スレッド生成済信号＝１？
YES
NO
S２
隣接プロセッサはフリー状態？
NO
YES
S３
プロセッサ状態テーブル更新
S５
当該フォーク要求を破棄する
フォーク先プロセッサへスレッド開始要求を送信し、要求プロセッサへフォーク応答を送信する
S４
エンド

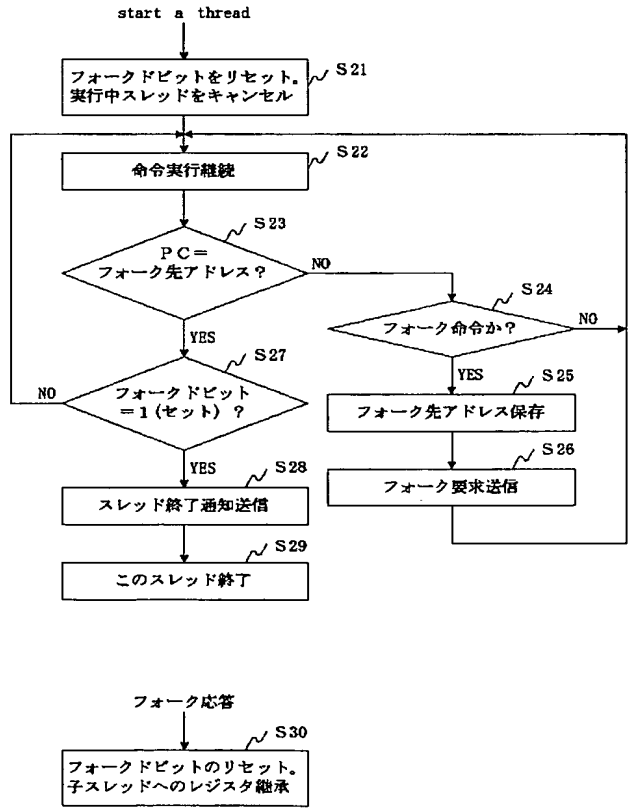
【図５】


【図５】
スレッド終了通知受信
S11
最古親スレッドか？
NO
YES
S14
最古親スレッドになるまでスレッド終了許可を保留する
S12
プロセッサ状態テーブル更新
S13
スレッド終了許可を送信する
エンド

【図６】

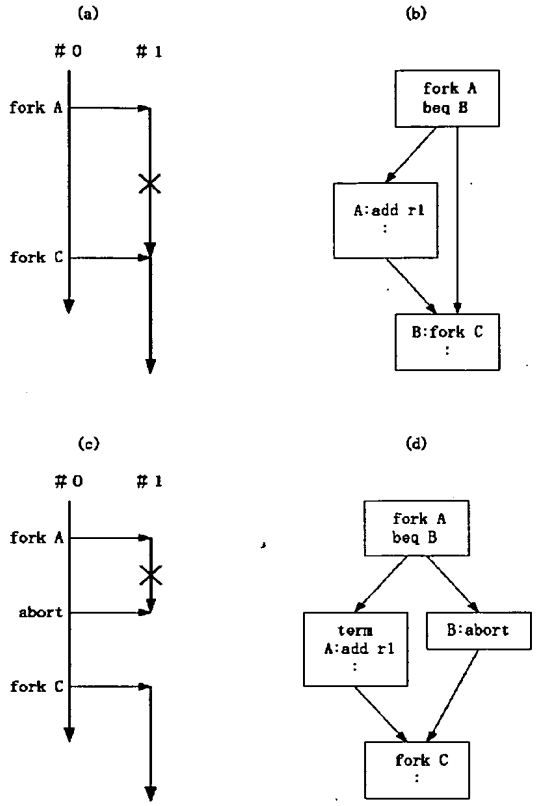

【図６】
fork address
7a
7b
7d 7e
start address
7c
21
PC
27
レジスタ
28
22
instruction fetch
31
30
29
23
shared memory
temporary buffer
execution unit
24-0
24-1
24-m
25
26
レジスタ転送ユニット

【図７】


【図７】
start a thread
フォークビットをリセット。実行中スレッドをキャンセル
S21
命令実行継続
S22
S23
ＰＣ＝フォーク先アドレス？
NO
YES
S24
フォーク命令か？
NO
YES
S27
フォークビット＝１(セット)？
NO
YES
S25
フォーク先アドレス保存
S26
フォーク要求送信
S28
スレッド終了通知送信
S29
このスレッド終了
フォーク応答
S30
フォークビットのリセット。子スレッドへのレジスタ継承

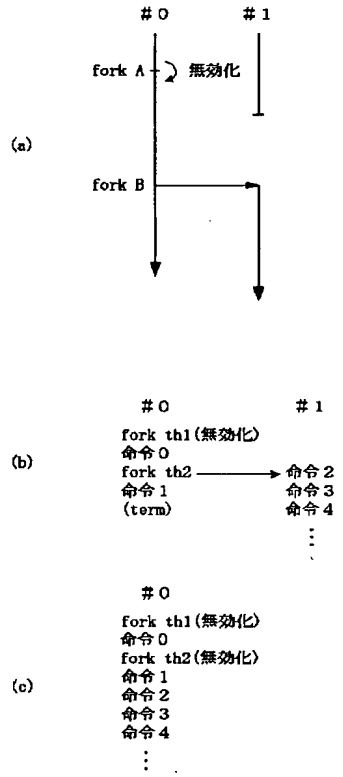
【図８】

【図8】



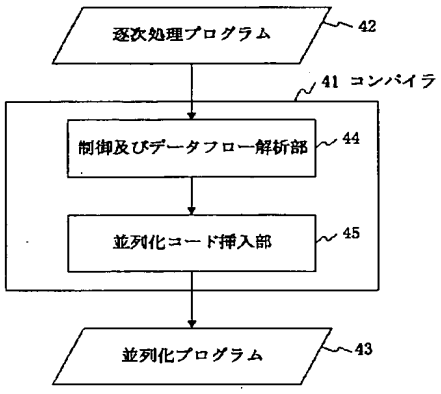
【図９】

【図9】



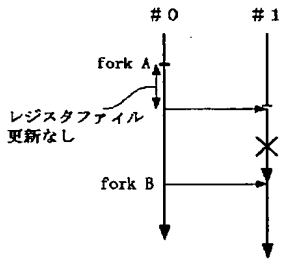
【図１０】

【図10】



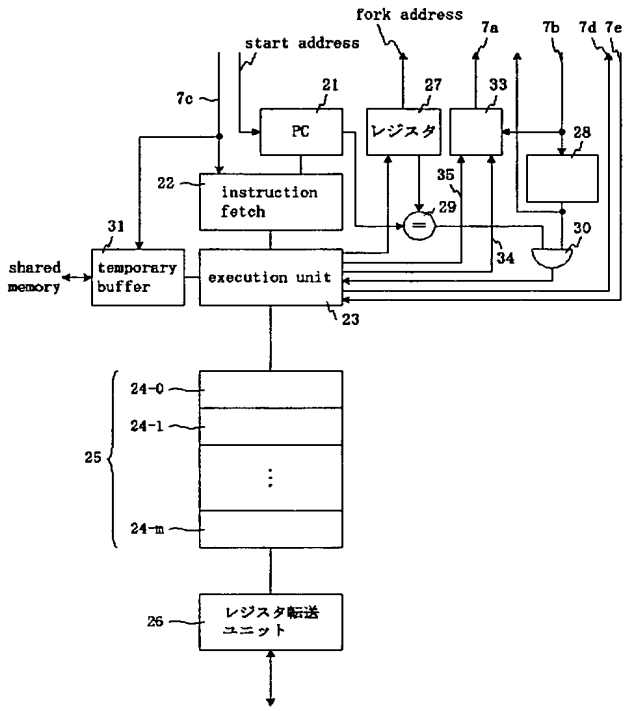
【図１３】

【図13】



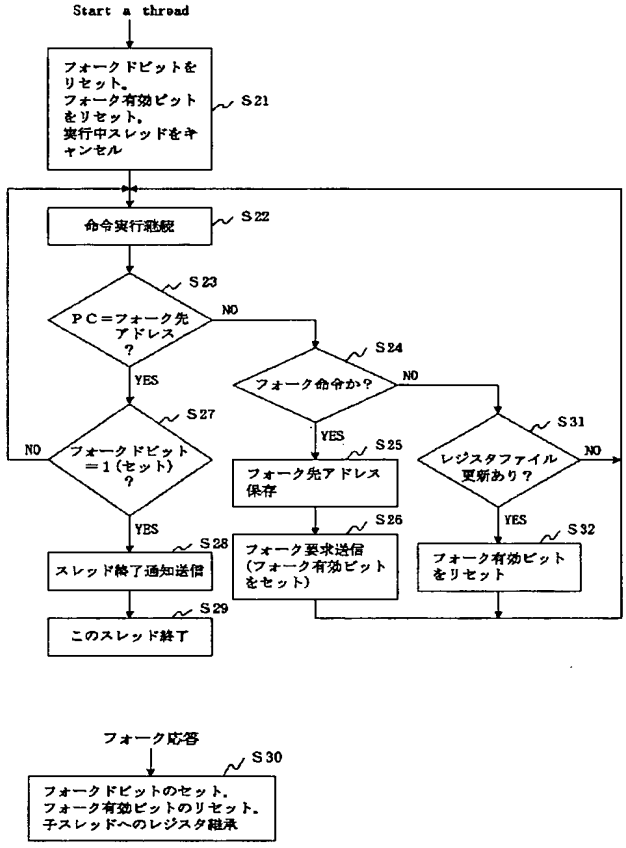
【図１１】

【図11】



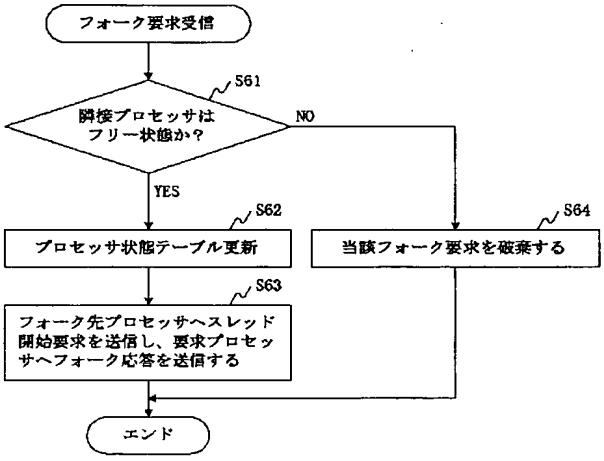
【図１２】

【図12】



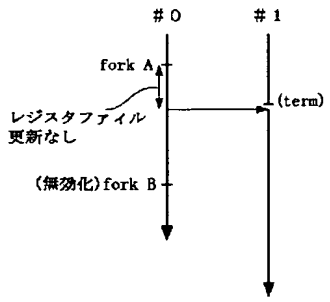
【図２２】

【図22】



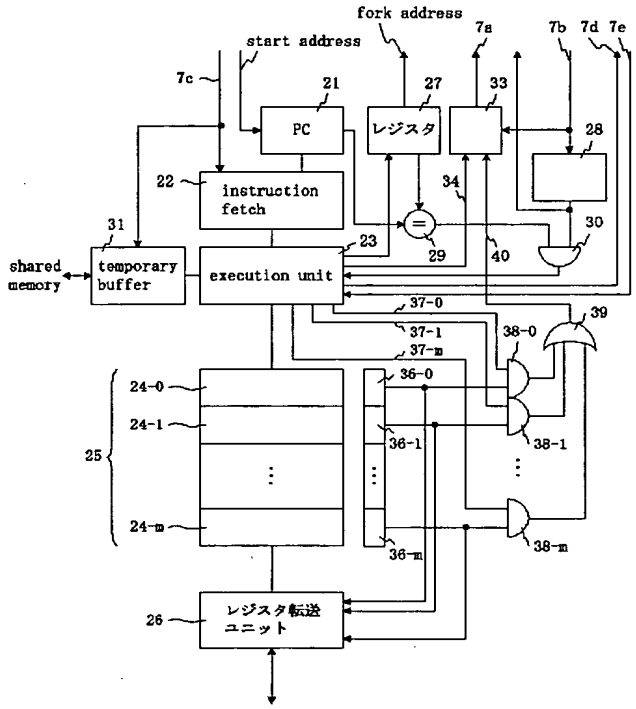
【図２８】

【図28】



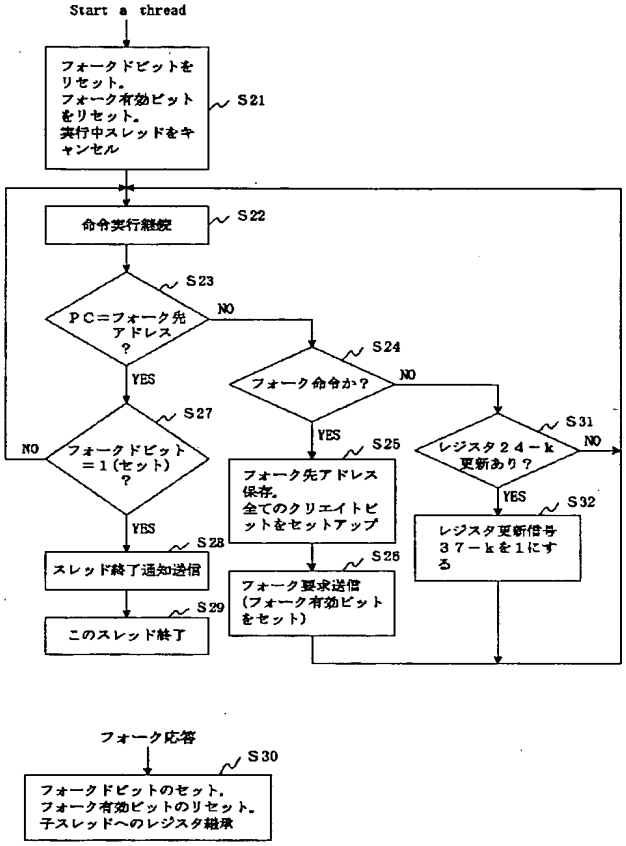
【図１４】

【図１４】



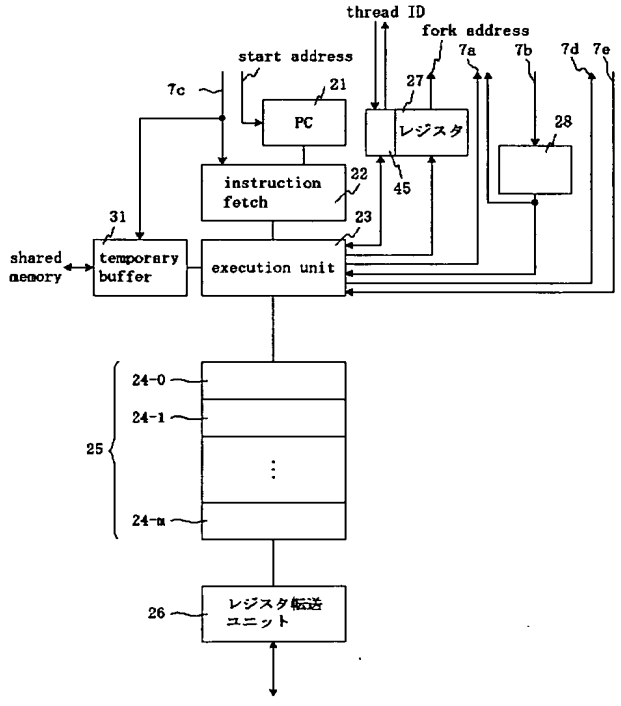
【図１５】

【図１５】



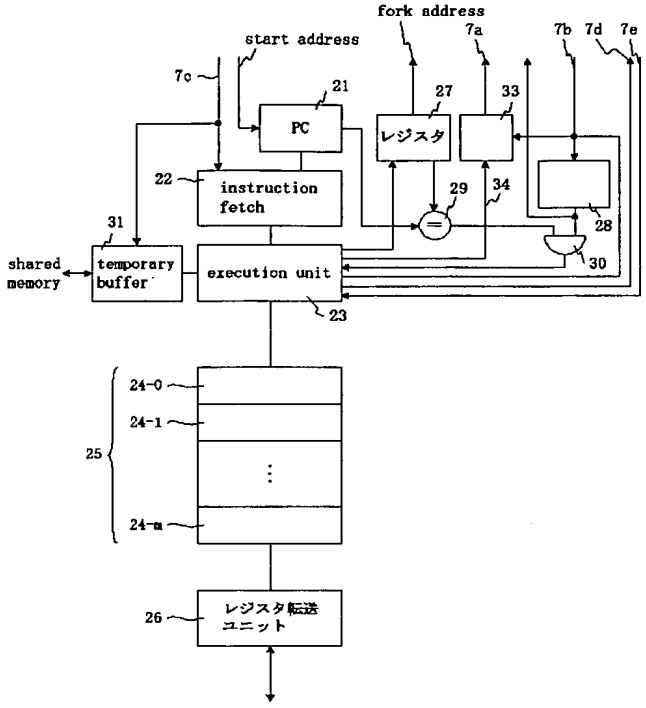
【図２０】

【図２０】



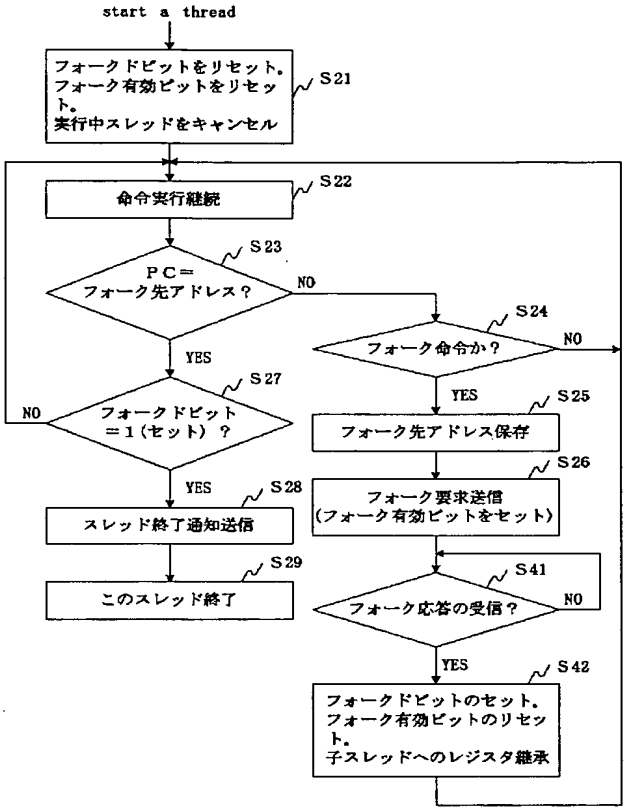
【図１６】

【図１６】



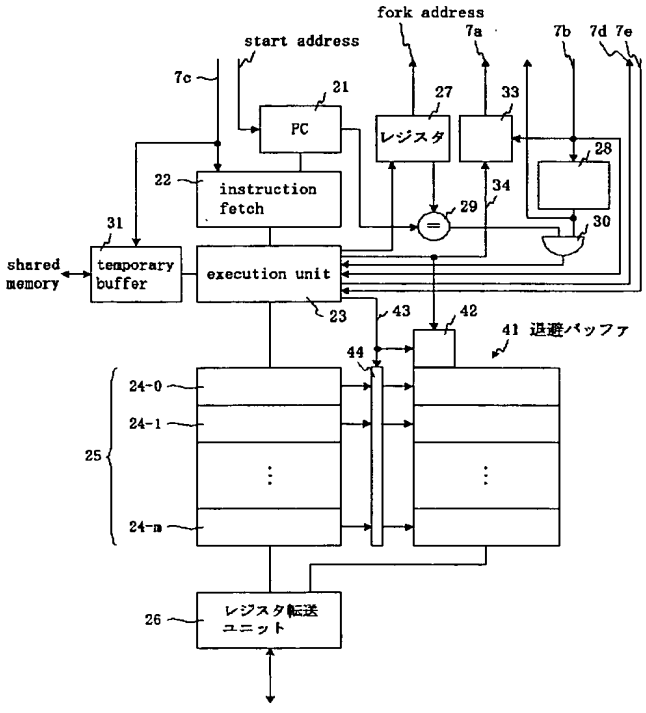
【図１７】

【図１７】

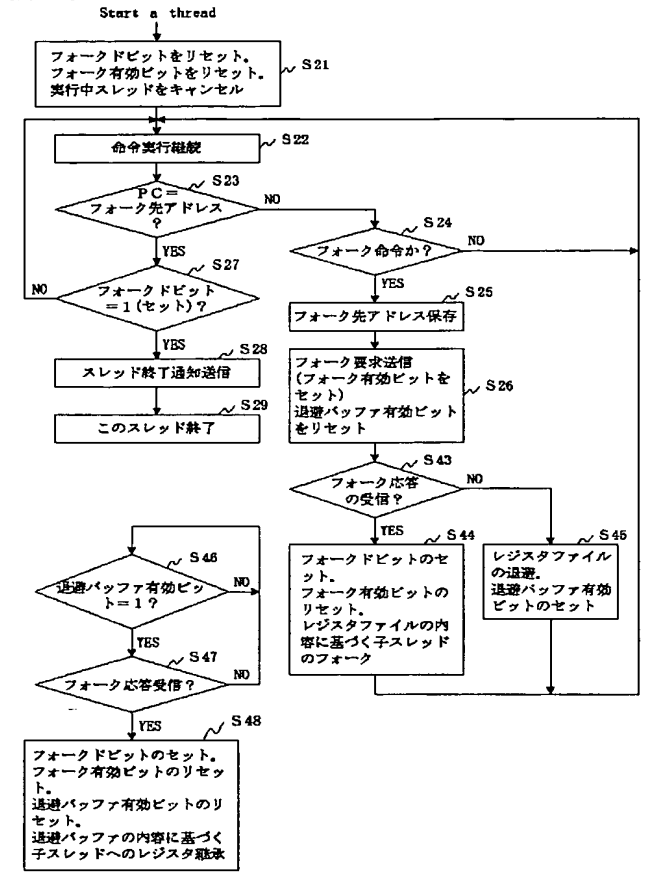


【図１８】

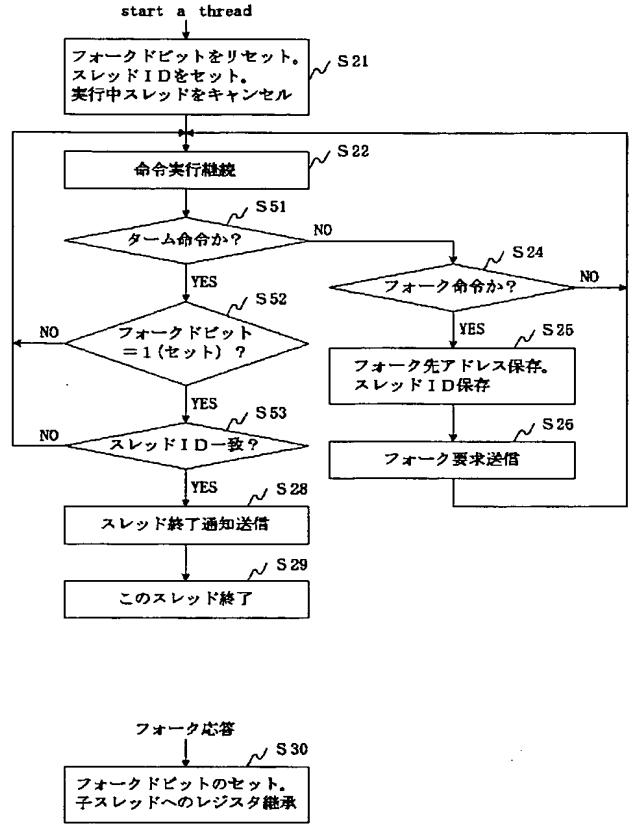
【図１８】



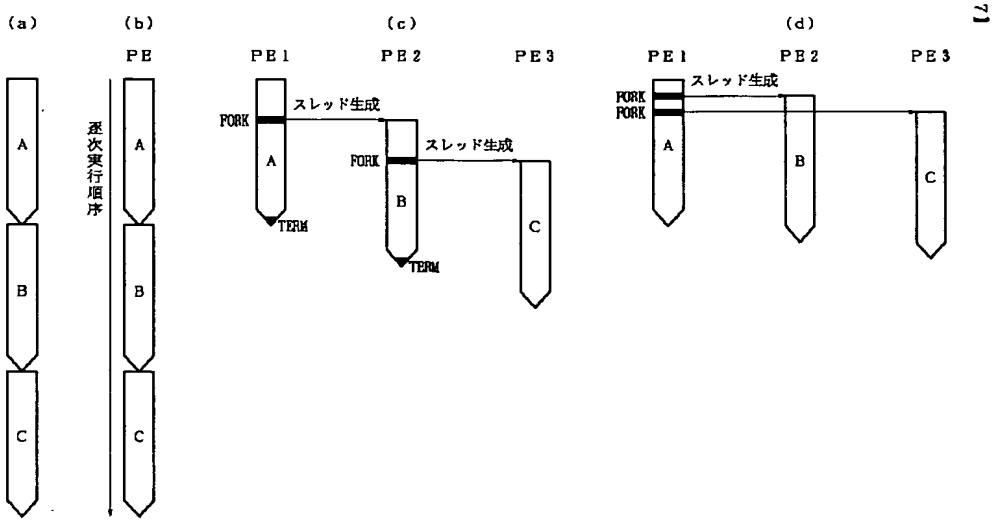
【図１９】


Start a thread
フォークビットをリセット。
フォーク有効ビットをリセット。
実行中スレッドをキャンセル
S21
命令実行継続
S22
PC＝
フォーク先アドレス
？
S23
NO
YES
フォーク命令か？
S24
NO
YES
フォークビット
＝１(セット)？
S27
NO
YES
フォーク先アドレス保存
S25
スレッド終了通知送信
S28
フォーク要求送信
(フォーク有効ビットを
セット)
退避バッファ有効ビット
をリセット
S26
このスレッド終了
S29
フォーク応答
の受信？
S43
NO
YES
退避バッファ有効ビット＝１？
S46
NO
YES
フォークビットのセット。
フォーク有効ビットのリセット。
レジスタファイルの内
容に基づく子スレッド
のフォーク
S44
レジスタファイル
の退避。
退避バッファ有効
ビットのセット
S45
フォーク応答受信？
S47
NO
YES
フォークビットのセット。
フォーク有効ビットのリセット。
退避バッファ有効ビットのリセット。
退避バッファの内容に基づく
子スレッドへのレジスタ継承
S48

【図２１】

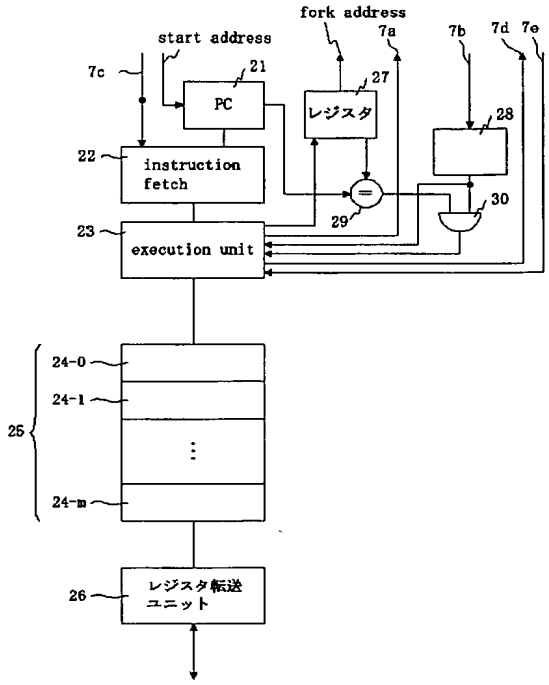

start a thread
フォークビットをリセット。
スレッドＩＤをセット。
実行中スレッドをキャンセル
S21
命令実行継続
S22
ターム命令か？
S51
NO
YES
フォークビット
＝１(セット)？
S52
NO
YES
スレッドＩＤ一致？
S53
NO
YES
スレッド終了通知送信
S28
このスレッド終了
S29
フォーク命令か？
S24
NO
YES
フォーク先アドレス保存。
スレッドＩＤ保存
S25
フォーク要求送信
S26
フォーク応答
フォークビットのセット。
子スレッドへのレジスタ継承
S30

【図３７】


(a)
(b)
PE
A
B
C
逐次実行順序
(c)
PE1
PE2
PE3
FORK
スレッド生成
TERM
(d)

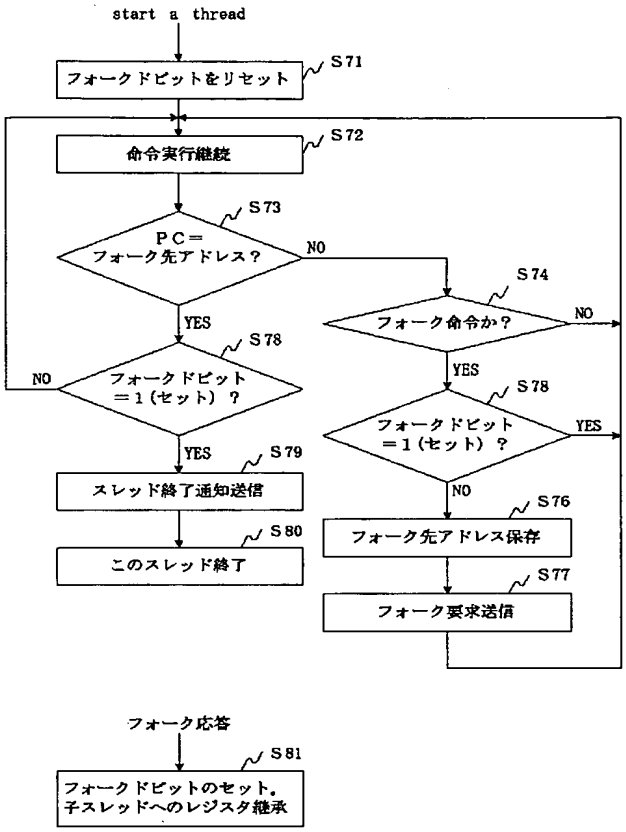
【図２３】

【図２３】



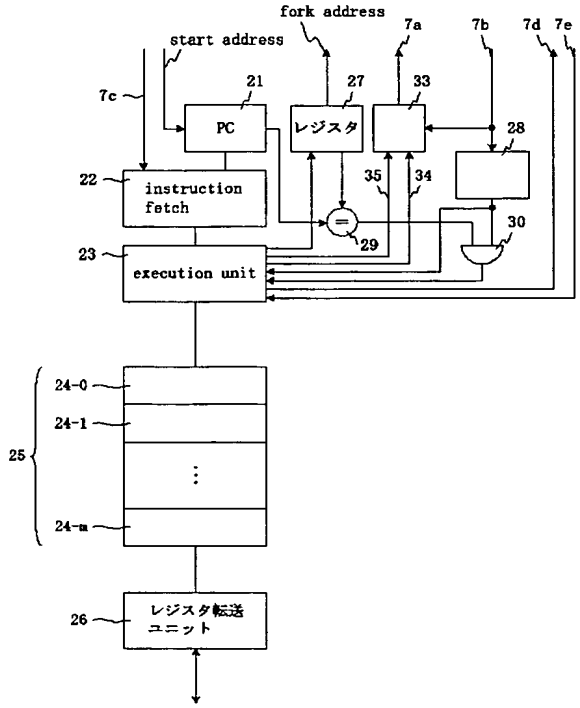
【図２４】

【図２４】



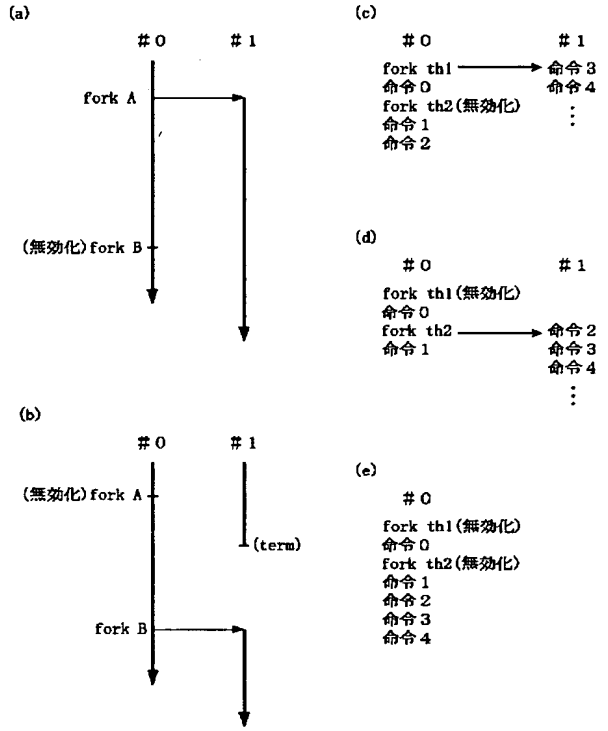
【図２６】

【図２６】



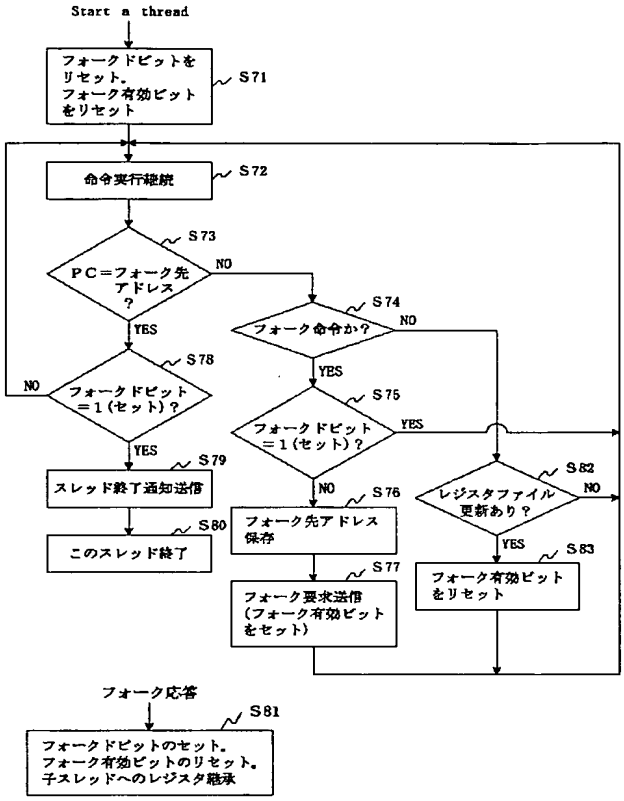
【図２５】

【図２５】



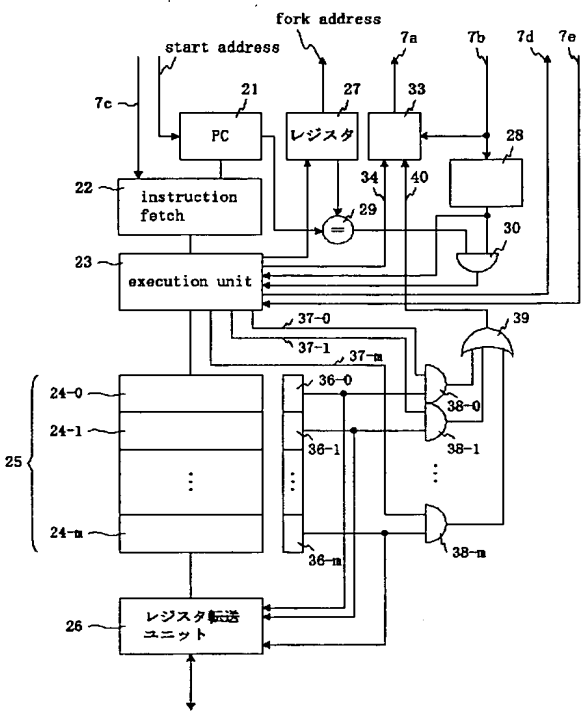
【図２７】

【図２７】



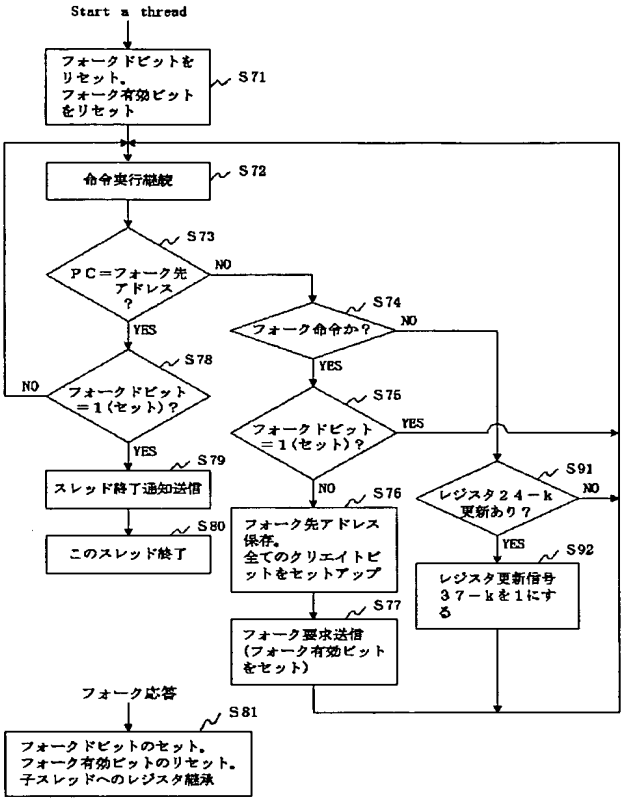
【図２９】

【図２９】



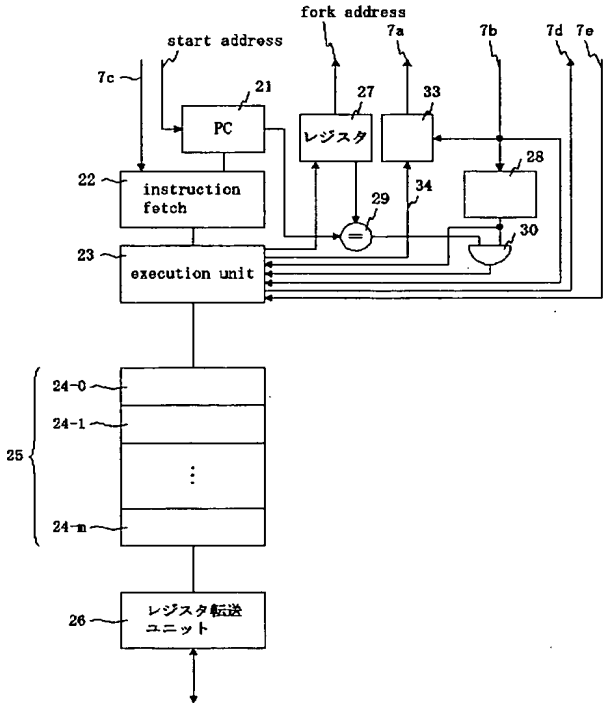
【図３０】

【図３０】

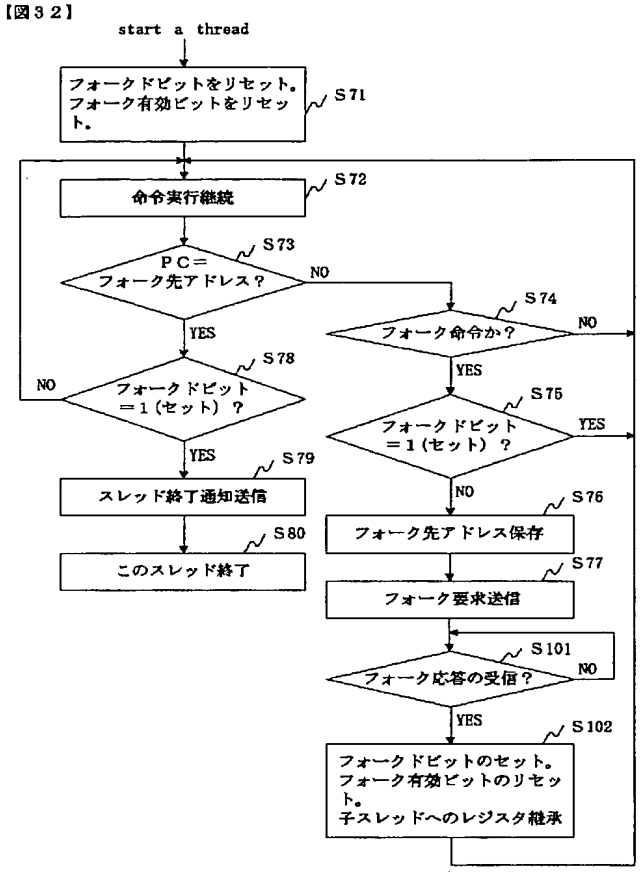


【図３１】

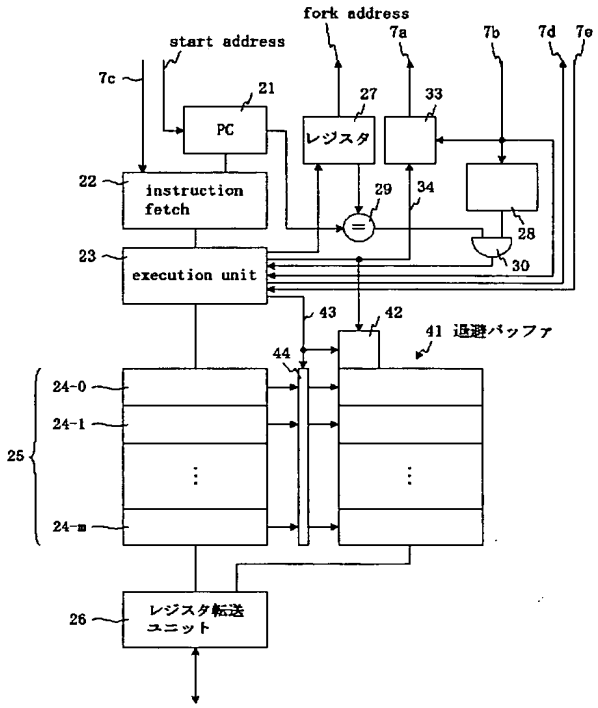
【図３１】



【図３２】

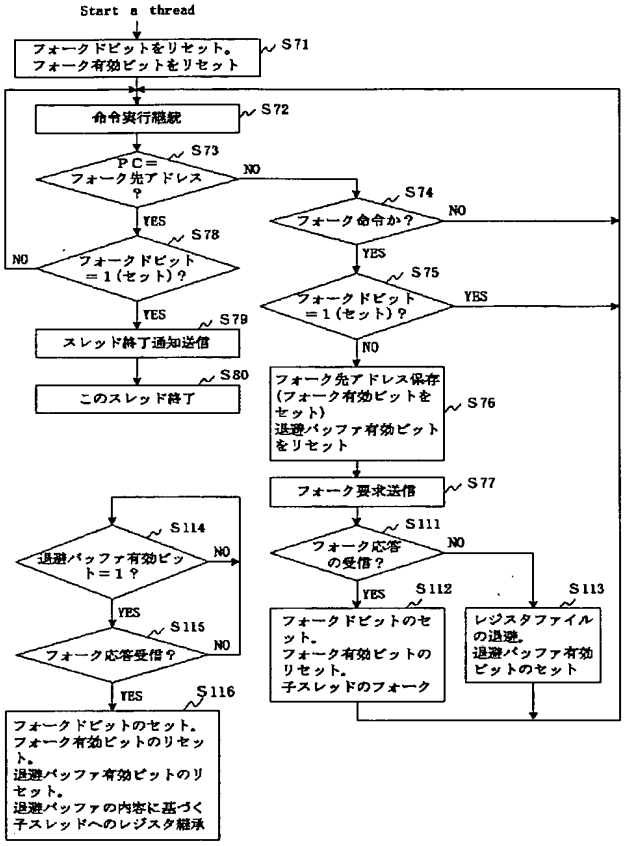

start a thread
フォークビットをリセット。
フォーク有効ビットをリセット。
S71
命令実行継続
S72
PC＝
フォーク先アドレス？
S73
NO
YES
フォークビット
＝1（セット）？
S78
NO
YES
スレッド終了通知送信
S79
このスレッド終了
S80
フォーク命令か？
S74
NO
YES
フォークビット
＝1（セット）？
S75
YES
NO
フォーク先アドレス保存
S76
フォーク要求送信
S77
フォーク応答の受信？
S101
NO
YES
フォークビットのセット。
フォーク有効ビットのリセット。
子スレッドへのレジスタ継承
S102

【図３３】


fork address
start address
7a
7b
7d
7e
7c
21
PC
27
レジスタ
33
22
instruction fetch
29
34
28
30
23
execution unit
43
42
41 退避バッファ
44
24-0
24-1
24-m
25
26
レジスタ転送ユニット

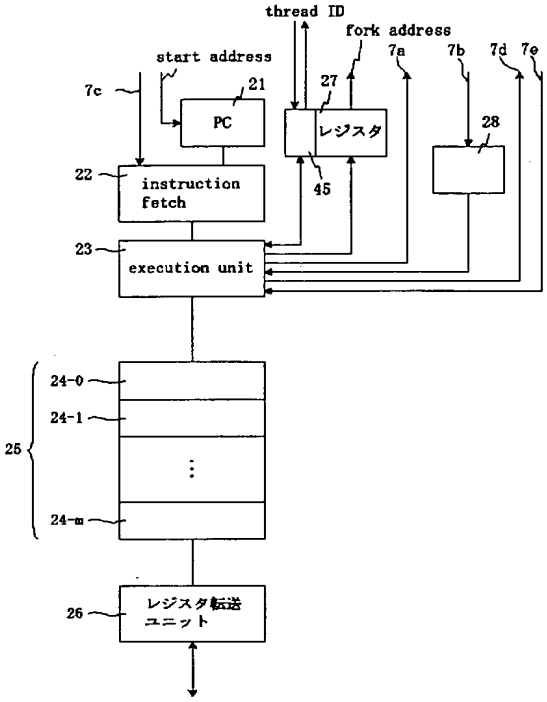
【図３４】

【図３４】

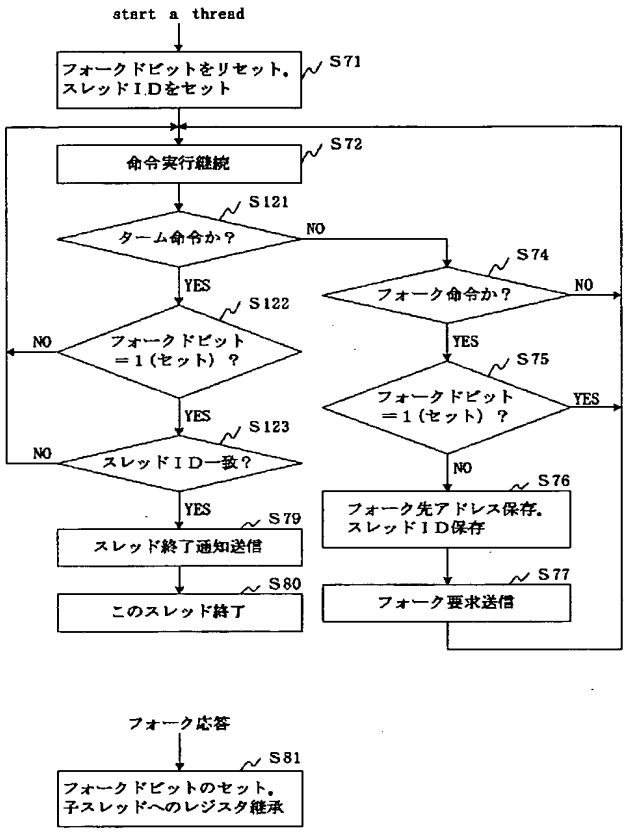


【図３５】

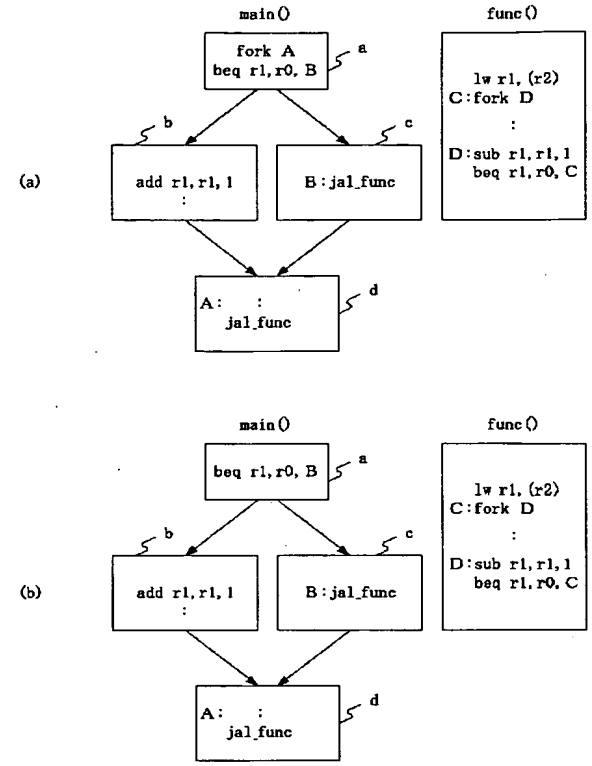
【図３５】



【図３６】


start a thread
フォークドビットをリセット，
スレッドI.Dをセット
S71
命令実行継続
S72
S121
ターム命令か？
NO
YES
S74
フォーク命令か？
NO
YES
S122
フォークドビット
＝1（セット）？
NO
YES
S75
フォークドビット
＝1（セット）？
YES
NO
S123
スレッドＩＤ一致？
NO
YES
S76
フォーク先アドレス保存，
スレッドＩＤ保存
S79
スレッド終了通知送信
S77
フォーク要求送信
S80
このスレッド終了
フォーク応答
S81
フォークドビットのセット，
子スレッドへのレジスタ継承

【図３８】


(a)
main()
fork A
beq r1,r0, B
a
b
c
add r1,r1,1
:
B:jal_func
A: :
jal_func
d
func()
lw r1, (r2)
C:fork D
:
D:sub r1,r1,1
beq r1,r0, C
(b)
main()
beq r1,r0, B
a
b
c
add r1,r1,1
:
B:jal_func
A: :
jal_func
d
func()
lw r1, (r2)
C:fork D
:
D:sub r1,r1,1
beq r1,r0, C